

EDITIONS DU P.S.I.

la découverte de l'
ORIC
(ORIC-1 et ATMOS)

daniel-jean david

La découverte de L'ORIC-1 et ATMOS

La collection « **MATERIELS** » s'intéresse à l'utilisation de tel ou tel type d'ordinateur, depuis la première prise de contact jusqu'à l'utilisation la plus pointue.

— La découverte de l'Apple II — Frédéric Lévy et Dominique Schraen
— Exercices pour Apple II — Frédéric Lévy
— La pratique de l'Apple II — tomes 1 et 2 — Nicole Bréaud-Pouliquen
— La pratique de l'Apple II — tome 3 — Nicole Bréaud-Pouliquen et Daniel-Jean David
— La découverte du Commodore 64 — Daniel-Jean David
— La pratique du Commodore 64 — tome 1 — Daniel-Jean David
— La découverte du Dragon — Frédéric Lévy et Dominique Schraen
— Dragon tout feu tout flammes — Trévor Toms et John Phipps — traduit par Olivier Arnaud
— La découverte du FX-702 P — Jean-Pierre Richard
— Exercices pour Goupil — Yves Martin
— Programmer HP-41 — Philippe Deschamps et Jean-Jacques Dhénin
— La découverte du PB-100 — Pierrick Moigneau
— La découverte du PC-1251 — Jean-Pierre Richard
— La découverte du PC-1500 — Jean-Pierre Richard
— Le petit livre du Spectrum — Trévor Toms — traduit par Allan Keil
— Exercices pour Spectrum — Julien Lévy
— La pratique du Spectrum — tome 1 — Xavier Linant de Bellefonds
— La pratique du Spectrum — tome 2 — Marcel Henrot
— La découverte du T07 — Dominique Schraen et Maurice Charbit
— Exercices pour TRS-80 — Frédéric Lévy
— Les graphiques sur TRS-80 — Donn Inman — traduit par Alain Pinaud
— La découverte de la TI 57 — Xavier de la Tullaye
— La découverte du TI-99/4A — Frédéric Lévy et Dominique Schraen
— Exercices pour TI-99/4A — Frédéric Lévy
— La découverte du VIC — Daniel-Jean David
— La pratique du VIC — tome 1 — Daniel-Jean David
— Le petit livre du ZX-81 — Trévor Toms — traduit par Ghislaine Lapeyre
— La pratique du ZX-81 — tome 1 — Xavier Linant de Bellefonds
— La pratique du ZX-81 — tome 2 — Marcel Henrot
— La découverte de l'Oric — Daniel-Jean David
— La découverte de l'Alice et MC/10 — Maurice Charbit

Autres ouvrages relatifs à l'ORIC-1 :

— ORIC pour tous — Jacques Boisgontier et Sophie Brébion
— 52 programmes pour tous, ORIC — Jacques Boisgontier

RAPPELS

Les séries :

En fait, il faudrait parler de niveaux, puisque la couleur attachée à chaque ouvrage permet de situer la « force » de celui-ci selon le code suivant :

Série VERTE : ouvrage d'initiation ne nécessitant que des connaissances de base.
Série BLEUE : suppose une connaissance élémentaire du sujet traité.
Série ROUGE : ouvrage d'approfondissement, niveau de complexité moyen.
Série NOIRE : ouvrage d'approfondissement, niveau de complexité élevé.

Les collections :

Les ouvrages d'Edition du PSI, actuellement une centaine, sont répartis en collections :

« LANGAGES », « MATERIELS », « PROGRAMMES », « GUIDES PRATIQUES », « MEMENTOS », « UTILISATIONS DE L'ORDINATEUR », « LOGIGUIDES » et pour l'initiation, outre quelques livres hors collection, « **... POUR TOUS** ».

ISBN : 2-86595-109-X

La découverte de L'ORIC-1 et ATMOS

par
Daniel-Jean David



Editions du P.S.I.
1984

Daniel-Jean DAVID *enseigne l'informatique de gestion à l'Université de Paris-1 Panthéon Sorbonne.*

Par ailleurs, il enseigne l'utilisation des microprocesseurs à l'E.N.S.A.M.

Ses sujets de recherche vont de l'informatique graphique aux techniques d'interface des microprocesseurs et des systèmes multi-processeurs.

Spécialiste du microprocesseur 6502, il a donné, à Paris, de nombreux séminaires sur les microprocesseurs, le KIM, le SYM et le P.E.T.

Il est directeur de la publication de "La Commode", revue trimestrielle consacrée aux ordinateurs Commodore, Atari et Oric.

S O M M A I R E

PRESENTATION

Ce livre a pour but de vous permettre de tirer le meilleur parti possible de votre micro-ordinateur ORIC. Après une introduction formée de rappels généraux sur l'informatique, il comprend essentiellement une introduction progressive au langage Basic, qui est le langage de programmation utilisé le plus souvent sur l'ORIC.

Bien entendu, on y exploite au maximum les particularités de l'ORIC. La structure de cette partie est conçue pour permettre l'acquisition progressive des connaissances : elle est formée de chapitres, ou plutôt de séries.

Dans chaque série, on bâtit petit à petit un programme, par variations continues, en introduisant peu à peu les notions nouvelles.

Il est recommandé au lecteur de bien suivre cette progression, d'essayer réellement sur son ORIC les différentes versions des programmes, et même d'en imaginer d'autres.

C'est la condition nécessaire d'acquisition de connaissances durables. Toutefois, ce livre devrait aussi permettre aux personnes qui n'ont pas encore d'ORIC de se faire une idée des possibilités de cet ordinateur.

Enfin, l'ouvrage se termine par des annexes où sont fournies des informations de référence : explication de chaque instruction Basic, messages d'erreurs, points particuliers traités sous forme de questions et de réponses.

Dernière précision : tous les programmes cités ont été réellement essayés au clavier d'un ORIC.

Les exemples de ce livre qui sont purement en Basic sont aussi valables pour l'ATMOS.

CHAPITRE I

PRISE DE CONTACT

LE MESSAGE "47870 BYTES FREE" BITS, OCTETS, INFORMATIONS

A peine vous êtes-vous assis devant votre ORIC et l'avez-vous mis sous tension qu'il affiche "47870 BYTES FREE"(1) dont la traduction est : "47870 OCTETS LIBRES". Que sont donc ces "OCTETS" dont on vous annonce qu'ils sont "LIBRES" ?

Pour comprendre - et nous nous en excusons - il nous faut voir un certain nombre de notions. Nous essaierons d'être brefs. L'ORIC est un ordinateur, c'est-à-dire une machine de traitement automatique des informations. On conçoit que les ordinateurs aient un champ d'applications extrêmement vaste puisque, en définitive, toute activité humaine se ramène à un certain traitement d'informations.

Une des opérations essentielles que l'ordinateur doit pouvoir effectuer sur une information est de la mémoriser, pour être capable de l'utiliser à différents moments. Mais la mémorisation de l'information implique sa mise sous une forme physique convenable pour pouvoir être stockée dans les circuits de l'ordinateur.

---

(1) Le nombre indiqué est plus petit sur le modèle 16K (15103).

Dans l'état actuel de la technologie, le seul codage pratique est de type binaire, car on sait très bien réaliser des éléments capables de prendre deux états bien distincts, par exemple : présence ou absence d'un trou sur une carte, élément de bande magnétique aimanté dans un sens ou dans l'autre, élément de circuit électronique porté au potentiel 5 V, ou restant à 0 etc.

Un tel élément qui, en tant qu'information est en somme capable de contenir la réponse par oui ou par non à une certaine question s'appelle un *BIT* (abréviation anglaise de "chiffre binaire" : Binary Digit).

Mais un seul bit forme le plus souvent une information trop élémentaire à manipuler de façon pratique. C'est pourquoi on manipule généralement des groupes de bits. Le groupe le plus souvent envisagé est le groupe de 8 bits ou *OCTET*. Pour comprendre ce que signifie "en avoir 47870 de libres", il nous faut maintenant voir ce que l'on peut représenter avec un octet, c'est-à-dire quelles informations peuvent être enfermées dans un octet.

Pour visualiser un octet sur le papier, il nous faut introduire deux symboles correspondant aux deux états que peut prendre chaque bit de l'octet. Si nous prenons pour symboles 0 et 1, un octet pourra être, par exemple, 01000001 ou 10100101 ou encore 01101001. Notons tout de suite que, comme on a deux possibilités pour chaque bit, on a 2^8 = 256 possibilités différentes pour un octet.

Maintenant, si nous voulons qu'un octet représente un nombre, c'est très facile. Il suffit de considérer que l'on a exprimé le nombre dans le système de numération binaire (à base 2). Ainsi, par exemple, 01000001 vaudra $1 + 0x2 + 0x2^2 + 0x2^3 + 0x2^4 + 0x2^5 + 1x2^6 + 0x2^7 =$ $1 + 64 = 65$. De la même façon qu'en décimal 1702 vaut $2 + 0x10 + 7x10^2 + 1x10^3$. Chaque bit représente un chiffre du système binaire, d'où son nom. Le nombre le plus petit que l'on puisse représenter est 00000000 (0) ; le plus grand est 11111111 (255) : on retrouve les 256 combinaisons. On voit aussi une chose : comme on veut pouvoir manipuler des nombres plus grands que 255, il faudra quelquefois que les nombres occupent plusieurs octets.

Peut-on ranger autre chose que des nombres dans un octet ? Bien sûr ! Supposons qu'on décide que 01000001 = A, 01000010 = B etc. pour toutes les lettres. Là encore, on peut avoir un jeu de 256 caractères différents, ce qui permet les lettres majuscules et minuscules, les chiffres, les caractères de ponctuation et bien d'autres.

On pourra alors stocker n'importe quel texte dans la mémoire, à raison d'un caractère par octet.

On peut donc ranger un texte de 47000 caractères dans la mémoire de l'ORIC, soit trente pages de ce livre. Cela vous donne maintenant une meilleure idée de ce que nous avons comme mémoire.

Mais il y a encore une autre catégorie d'informations qui doivent entrer dans la mémoire, et c'est même une caractéristique fondamentale des ordinateurs. En effet, un ordinateur a besoin, pour fonctionner, d'instructions qui lui disent ce qu'il a à faire. Ces instructions convenablement codées résident en mémoire, au même titre que les informations à traiter ; de façon analogue, un employé qui effectue des calculs de comptabilité n'a-t-il pas en mémoire la liste des opérations qu'il doit effectuer à côté des chiffres qu'il doit manipuler ?

LA CONFIGURATION DE L'ORIC

La mémoire n'est qu'une partie de la configuration dont on dispose avec l'ORIC. Sur les gros ordinateurs, on distingue facilement, car ils occupent chacun une armoire, les éléments principaux qui sont l'unité centrale (où s'effectuent les traitements) et les périphériques qui servent à communiquer avec le monde extérieur (notamment saisir les données à traiter et fournir les résultats obtenus).

Les mêmes éléments existent sur l'ORIC. Extérieurement, on ne voit que les **périphériques** :

- ***Le clavier*** : qui va nous servir à entrer des données et des instructions ;

- ***L'écran de télévision*** : sur lequel s'afficheront les résultats (28 lignes de 38 caractères). Le couple clavier/écran est l'instrument du dialogue entre vous et votre machine ;

- ***L'unité de cassettes magnétiques*** : A la différence des périphériques de communication précédents, il s'agit plutôt d'un périphérique de stockage ou mémoire de masse, qui permet de stocker des programmes ou des données si la mémoire centrale est insuffisante. Mais la cassette est aussi un périphérique de communication entre ORIC : vous pouvez envoyer un programme ou des données sur cassettes à un ami, qui possède aussi un ORIC.

L'unité centrale est cachée, mais elle existe et elle est en fait formée de deux éléments : *le processeur et la mémoire centrale.*

- ***Le processeur*** est ici un ***microprocesseur***, c'est-à-dire un circuit intégré à grande échelle capable, à lui seul, de commander tout le système : il cherche en mémoire les instructions successives, les interprète et les exécute ; il envoie aux autres composants du système les ordres nécessaires. Dans le cas de l'ORIC, le microprocesseur est un 6502 de MOS Technology, un des plus efficaces du marché.

- ***La mémoire.*** Les 47000 octets vus précédemment n'en sont qu'une partie. C'est en fait 80000 octets de mémoire qui sont présents. (Les informaticiens disent 80K où $K = 2^{10} = 1024$). Parmi les 80K, 16K sont de la *ROM* (Read Only Memory), mémoire écrite une fois pour toutes, encore appellée Mémoire Morte ou *MEM*, qui contient les programmes invariables permettant à l'ORIC de se mettre au service de l'utilisateur (c'est le système d'exploitation que nous présentons plus en détail à la section suivante). Les 64K restants sont de la *RAM* (Random Access Memory), c'est-à-dire de la mémoire que l'on peut lire ou écrire (ou Mémoire Vive *MEV*). Elle contient les programmes de l'utilisateur et les données variables. Seuls 48K en sont accessibles et l'ORIC en réserve 1K pour son usage, d'où les 47K restant libres pour l'utilisateur. Il faut encore en retirer 1K ou 8K selon le mode d'affichage basse ou haute résolution, pour la mémoire d'écran.

La **figure 1** donne une vue synoptique de la configuration de l'ORIC. Nous avons maintenant complètement interprété "47870 BYTES FREE". Il nous faut maintenant expliquer la ligne "ORIC EXTENDED BASIC V 1.Ø" qui le précède. Ce sera le but de la section suivante.

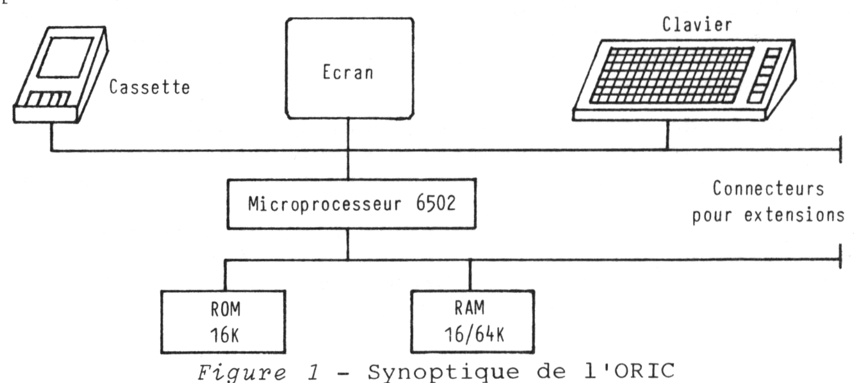


Figure 1 - Synoptique de l'ORIC

PROGRAMMES - SYSTEME D'EXPLOITATION - BASIC

On vient de voir que le microprocesseur, pièce maîtresse de l'unité centrale était capable de chercher en mémoire les *instructions* successives, de les décoder et d'y obéir. De fait, il n'est capable que de cela ! Ensuite, pour obtenir quoi que ce soit de l'ordinateur, il faudra lui fournir les instructions convenables. Sans instructions précises, l'ordinateur ne sait rien faire, il n'a aucune initiative.

Une suite d'instructions que l'ordinateur doit exécuter successivement s'appelle un *programme*. Fournir une telle suite d'instructions permettant de résoudre un certain problème s'appelle programmer.

Vous concevez donc que vous allez devoir fournir des programmes à votre ORIC, que vous introduirez par le clavier. Mais, pour que l'ORIC prenne en compte ce que vous frappez au clavier, il faut qu'il ait un programme qui le lui ordonne. Il n'aurait pas tout seul l'initiative d'aller voir si quelqu'un tape sur le clavier...

Heureusement, nous n'avons pas à écrire ce programme. En effet, l'ORIC - comme d'ailleurs tout autre ordinateur - est livré avec un ensemble de programmes fondamentaux qui sont indispensables à son utilisation : programme qui lit le clavier, programme qui affiche sur l'écran, programme qui gère les cassettes, programme qui attend les instructions de l'utilisateur etc. L'ensemble de ces programmes s'appelle le *système d'exploitation*.

Ces programmes résident en ROM, afin d'être conservés en permanence pour être disponibles dès qu'on met l'ORIC sous tension. Le message qui s'affiche sur l'écran dès cette mise sous tension est une manière de dire à l'utilisateur que le système d'exploitation se met à sa disposition et attend ses ordres.

Une composante très importante du système d'exploitation de l'ORIC est *l'interpréteur Basic* qui va nous permettre de fournir des instructions à l'ORIC sous une forme commode pour nous.

La question se pose en effet de savoir sous quelle forme - ou en quel langage - nous devons fournir nos instructions à l'ORIC. A dire vrai, un ordinateur ne "comprend" qu'un seul langage, le *langage-machine* ou *binaire*.

Par exemple, en langage-machine de l'ORIC, "ajouter 2 et 3" se dirait 10101001 00000010 00011000 01101001 000 00011 10000101 01010000 ... C'est extrêmement compliqué à comprendre et à utiliser pour l'homme ! C'est pourquoi d'autres langages de programmation ont été inventés, qui s'appellent langages évolués et sont plus proches du langage humain, plus proches des notations mathématiques usuelles, et donc plus faciles à utiliser. Dans un tel langage, "ajouter 2 et 3" se dirait, par exemple, A = 2+3 ; c'est beaucoup plus compréhensible, n'est-ce pas ?

Basic est l'un de ces langages évolués, le plus répandu à l'heure actuelle sur les PSI (Petits Systèmes Individuels). C'est probablement le plus simple à utiliser pour les débutants : son nom est l'abréviation de Beginners' All-purpose Symbolic Instruction Code = codage symbolique des instructions d'usage général pour les débutants.

C'est lui que nous utiliserons sur l'ORIC, mais nous ne pouvons le faire sans l'aide d'un programme du système d'exploitation. En effet, l'ORIC ne comprend vraiment que son langage-machine ; il faut donc un programme du système d'exploitation qui prenne chaque instruction de notre programme Basic et la traduise en Binaire pour l'exécuter : c'est le rôle - essentiel - joué par l'interpréteur Basic.

En résumé, l'affichage qui apparaît sur l'écran à la mise en route signifie : l'interpréteur Basic du système d'exploitation est à votre disposition - vous disposez pour votre programme Basic et vos données de 47870 octets libres. Le dernier mot "Ready" qui réapparaîtra à chaque fois que l'ORIC aura terminé une commande signifie "Je suis prêt et j'attends que vous tapiez la prochaine instruction".

Faisons-le...

LES DEUX MODES DE FONCTIONNEMENT DE BASIC

Mettons-nous au clavier et tapons :

BONJOUR ORIC 'Return'

(Tout message tapé par l'utilisateur se termine en principe par la touche 'Return' - retour chariot -. Dans la suite, nous cesserons progressivement de la faire figurer : elle sera sous-entendue et devra être tapée).

L'ORIC répond :

?SYNTAX ERROR

En effet, bien que notre message soit très poli, il ne forme pas une instruction Basic correcte, et par suite, l'ORIC (ou plutôt l'interpréteur Basic) la refuse.

Il ne faut pas faire de complexes à propos des messages d'erreur, mais chercher calmement l'erreur que l'on a faite ; le comportement de la machine est, dans tous les cas, parfaitement rationnel. L'ORIC a tout un répertoire de messages d'erreurs qui n'ont pas d'autre but que celui de nous aider à les corriger.

Tapons maintenant :

?"BONJOUR!JE SUIS L'ORIC" 'Return'

(les espaces s'obtiennent avec la barre d'espace).

Cette fois, on a tapé une bonne instruction, et on obtient comme réponse :

BONJOUR!JE SUIS L'ORIC
Ready

Vous n'obtenez pas cette réponse ? Etes-vous certain d'avoir bien tapé **tous** les caractères, y compris les guillemets ?

Le point d'interrogation signifie simplement "imprimer". On constate que l'on a obtenu une réponse immédiate à l'instruction.

De même, si on tape :

?2+2 'Return'

l'ORIC fera le calcul et imprimera immédiatement le résultat. On a donc un mode de fonctionnement à peu près semblable à celui d'une calculatrice de poche, où une instruction est exécutée dès qu'elle vient d'être tapée. On dit qu'il s'agit du "mode immédiat" ou "mode d'exécution immédiate" ou encore "mode direct".

L'ORIC a un second mode de fonctionnement. Tapons :

2Ø ?2+2 'Return'

Rien ne se produit. Tapons encore :

1Ø ?"BONJOUR" 'Return'

Toujours rien. Tapons :

RUN (les lettres R U N suivies de 'Return')

Cette fois, on obtient sur l'écran :

```
BONJOUR
4
Ready
```

Que s'est-il passé ? Eh bien, on a fonctionné dans le second mode, où les instructions ne sont pas exécutées immédiatement, mais mises en mémoire pour exécution ultérieure. Elles forment alors un programme qui est exécuté lorsque l'on tape la commande RUN en mode direct.

Sauf une ou deux exceptions, ce sont exactement les mêmes instructions qui peuvent être données en mode direct ou dans le second mode, qui s'appelle "mode différé" ou "mode programmé". Alors, à quoi reconnaît-on le mode ?

C'est très simple : en mode différé, toute instruction possède en tête un numéro de ligne, alors qu'en mode direct, il n'y a pas de numéro de ligne :

```
?2+2     mode direct
1Ø ?2+2  mode différé (il faut taper RUN pour
                        avoir la réponse).
```

En plus d'imposer le mode programmé, le numéro de ligne joue un autre rôle : on voit sur l'exemple précédent que l'instruction 10 a été exécutée avant l'instruction 20, bien qu'elle ait été tapée après : les instructions sont exécutées non pas dans l'ordre chronologique où elles ont été tapées, mais par ordre de numéros de ligne croissants.

L'ORIC est naturellement beaucoup plus utilisé en mode programmé, mais, avant, pour nous familiariser, nous allons effectuer quelques calculs arithmétiques en mode direct.

ARITHMETIQUE EN MODE DIRECT

Comme toute calculatrice, l'ORIC permet d'évaluer des expressions plus compliquées que 2+2 ! Dans tous les cas, on doit commencer par un ? ou par le mot PRINT dont il est l'abréviation : cela signifie "imprimez le résultat de l'expression qui suit".

Essayez (les 'Return' sont sous-entendus)

```
?5-3.5   (soustraction ; résultat 1.5)
?3*12    (multiplication ; résultat 36)
```

```
?1/3    (division ; résultat .333333333)
?2↑5    (élévation à la puissance ; résultat
         2⁵ = 32)
```

On voit donc quels sont les signes d'opération fondamentaux. Noter * et non x pour la multiplication, ↑ pour l'élévation à la puissance (on ne peut, au clavier, mettre un nombre plus haut que l'autre).

N.B. Le signe "élévation à la puissance" est listé ↑ à l'écran, mais il apparaît sous la forme ^ sur le clavier (shift 6).

VARIABLES

Etant donné une expression, on peut en faire autre chose que d'imprimer sa valeur : on peut mettre la valeur en mémoire pour utilisation ultérieure dans une autre expression.

Pour cela, il suffit de donner un nom à l'expression, en tapant par exemple :

```
A=2/3
```

On dit qu'on a constitué une variable de nom A. L'ORIC lui attribue automatiquement un emplacement mémoire (que vous n'avez pas à connaître : l'ORIC se charge entièrement de la gestion de la mémoire). Ensuite, le résultat est calculé et rangé dans la case mémoire considérée ; il n'apparaît pas sur l'écran.

La variable peut maintenant être utilisée dans une autre expression ; si on tape :

```
?2*A
```

on obtient :

```
1.33333333
```

Quels noms de variables peut-on prendre ? Les noms de variables sont pratiquement arbitraires, à ceci près qu'ils doivent se plier aux contraintes suivantes :

- *premier caractère :* lettre
- *caractères suivants :* lettres ou chiffres (ex. A, A1, VAR, RESULT, H2SO4).

L'ORIC permet plus de deux caractères, mais il ne prend en compte que les deux premiers. Si deux variables que vous considérez comme distinctes ont leurs deux

premiers caractères identiques, l'ORIC les confondra : par exemple, cas de ALBERT et ALAIN.

Le fait de pouvoir utiliser plus de deux caractères permet de choisir des noms "parlants" comme RESULT, TAUX, CAP..., mais le nom ne doit pas contenir un des mots particuliers au langage Basic, qu'on appelle les mots-clés : CHIFFRE est interdit, car il contient IF qui est un mot ayant une signification particulière pour Basic.

La liste des mots-clés "réservés" est fournie en annexe.

L'usage d'une variable ne détruit pas sa valeur, ou plutôt n'efface pas la mémoire correspondante. Ce n'est que lors d'une nouvelle instruction d'affectation (de la forme A=...) que la valeur sera changée.

Essayez le dialogue suivant : (nous omettons les 'Return' et les Ready) :

```
AL = 3
?AL+5
8            (5 + 3 = 8)
?3*AL        (AL vaut toujours 3)
9            (3 x 3 = 9)
ALAIN = 1    (ALAIN est la même variable que AL,
?AL+1         elle a maintenant pris la valeur 1)
2            (1 + 1 = 2)
```

Il reste encore une question : à combien de variables différentes avons-nous droit ? La limite réelle est en fait la taille mémoire, mais on peut espérer avoir plusieurs centaines de variables, c'est-à-dire de quoi traiter les problèmes les plus complexes.

EVALUATION DES EXPRESSIONS

On peut calculer des expressions plus compliquées, renfermant plusieurs opérations. Essayez :

```
?5+4*2
?2*3↑2
?5*3/4
```

La première a pour résultat 13, c'est-à-dire qu'on a effectué d'abord la multiplication. La deuxième donne 18, car on a d'abord effectué le 3^2. La troisième s'évalue en calculant d'abord 3 x 3 puis en divisant le résultat par 4.

En résumé, on effectue généralement de gauche à droite, mais on a une **règle de priorité** des opérateurs :

↑ est le plus prioritaire, puis on a :
- (prendre l'opposé, exemple : -X), puis :
* et / (ex-aequo), puis :
+ et - (ex-aequo).

Si l'on veut changer l'ordre de priorité, on emploie des parenthèses : un groupe entre parenthèses est toujours évalué en premier. Par exemple :

```
(2+10)/5  donne 2.4  alors que 2+10/5 donne 4
6-(3+5)   donne -2   alors que 6-3+5  donne 8
```

On peut avoir des parenthèses emboîtées, mais il faut toujours qu'il y ait le même nombre de parenthèses ouvrantes et fermantes et que l'expression ait un sens. Par exemple :

```
?(A-3*(B-C))/5↑(B*C)
```

On dispose, pour faciliter les calculs, d'un ensemble de fonctions mathématiques qui peuvent intervenir dans les expressions arithmétiques. Leur argument est, comme toute sous-expression entre parenthèses, évalué en priorité. La liste complète de ces fonctions est donnée en annexe, mais nous citons tout de suite : SIN (sinus), COS (cosinus), SQR (racine carrée), EXP (exponentielle)

```
?1+SQR(2)  donne 2.41421356
```

PRECISION DES NOMBRES

En examinant les résultats obtenus dans les exemples qui précèdent, nous pouvons faire un certain nombre de remarques :

- l'ORIC peut imprimer des nombres positifs ou négatifs. Il imprime un en tête pour un nombre négatif, rien pour un nombre négatif.

- l'ORIC peut imprimer des nombres entiers ou fractionnaires. Pour les nombres fractionnaires, on utilise un point (convention anglo-saxonne) au lieu de la virgule des Français. On utilise le système décimal, il n'y a pas à se soucier de binaire. Signalons aussi que les zéros sont barrés (Ø) pour ne pas être confondus avec la lettre O.

- L'ORIC imprime au maximum 9 chiffres significatifs. Il en utilise un peu plus dans la représentation interne pour faire les calculs, mais n'en "sort" que 9. La représentation interne est toujours fractionnaire, mais si, à la précision des calculs près, le nombre est assez proche d'un entier, il sera imprimé comme entier.

```
?14.99999999 donne 15
```

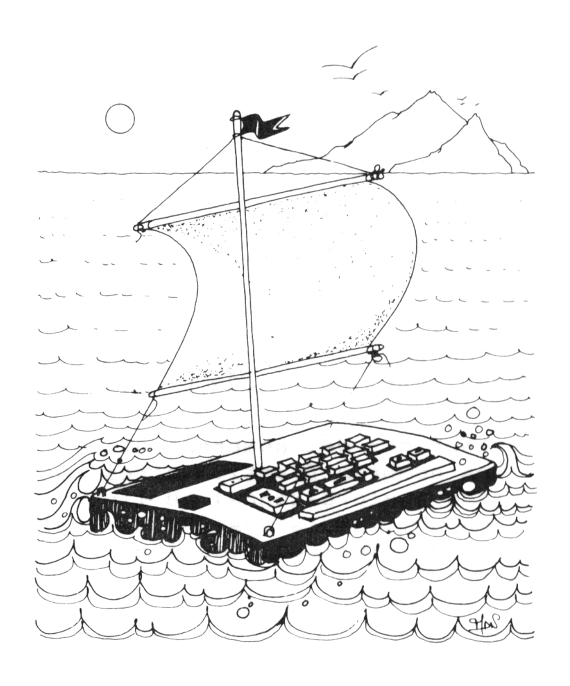
- si le nombre à imprimer est <0,1 ou >999999999, il est imprimé en "notation flottante", c'est-à-dire sous la forme : Sx.xxxxxxxxEstt (x et t sont des chiffres, S est le signe du nombre, s le signe de l'exposant).

?0.000123 donne 1.23E-04
ce qui se comprend comme $1,23 \times 10^{-4}$

?-1000000000 donne -1E+09
ce qui se comprend comme -1×10^{9}

Voilà terminée notre prise de contact avec l'ORIC. Nous espérons qu'elle n'a pas été trop ardue, et nous vous encourageons à essayer d'autres exemples, pour bien vous familiariser.

Nous sommes prêts maintenant à utiliser l'ORIC en mode programmé.



CHAPITRE II

INSTRUCTIONS FONDAMENTALES

LE PREMIER PROGRAMME

Tout en commençant par des notions très simples, nous passons maintenant à une programmation plus élaborée, en mode différé.

Tout traitement d'informations, donc tout programme, comprend trois étapes fondamentales, immuables :

- *l'acquisition* ou entrée des données à traiter ;
- *le traitement* des données c'est-à-dire le calcul des résultats ;
- *la sortie* des résultats.

On aura donc en Basic trois instructions fondamentales correspondant à ces trois actions ; on les retrouve dans le programme A qui a simplement pour but de calculer la surface d'un cercle de rayon R ($S = PI \times R^2$).

Programme A-1

```
10 INPUT R
20 S=PI*R↑2
30 PRINT S
```

Les programmes sont identifiés par une lettre suivie d'un numéro de version.

L'instruction 10 correspond à l'entrée au clavier du rayon R. Lorsque vous aurez tapé RUN, en exécution de

cette instruction, l'ORIC affichera sur l'écran un ? et le curseur clignotant. Il se mettra en attente que vous tapiez une valeur du rayon. Lorsque vous l'aurez fait, admettons que vous ayez tapé 1.5 (ce qui veut dire 1,5) sans oublier le 'Return', l'ORIC affectera cette valeur à la variable et passera à l'instruction suivante.

L'instruction 2Ø est l'instruction du calcul proprement dit de la surface à laquelle on a donné le nom de variable S. L'expression arithmétique Basic est une copie presque conforme (les signes d'opération sont obligatoires) de la formule mathématique qui intervient. Notons que le Basic de l'ORIC considère PI comme une variable réservée dont il "connaît" la valeur (3,14...).

L'instruction 3Ø demande simplement l'affichage du résultat 7.06858347, comme nous l'avons vu en mode direct.

Tout ceci est simple n'est-ce pas ? Eh bien cela renferme 90% de tout le Basic, puisque tout repose sur les trois opérations fondamentales que nous avons citées.

Bien que simple, le programme que nous venons d'écrire fonctionne de façon satisfaisante : pour calculer la surface d'un cercle, on tape RUN puis, dès que l'ORIC a affiché un point d'interrogation, on tape la valeur du rayon considéré, et dès qu'on a appuyé sur la touche 'Return', le résultat apparaît sur l'écran. Et à chaque fois qu'on tape RUN, le programme est réexécuté pour un nouveau rayon : on peut avoir la surface correspondant à autant de rayons que l'on veut. Le fait d'exécuter le programme ne l'"use" pas, c'est-à-dire qu'il reste en mémoire et, dès qu'on tape la commande directe RUN suivie de 'Return', le programme présent en mémoire s'exécute.

Quelques perfectionnements

Toutefois, le comportement de ce programme a quelques petits inconvénients auxquels nous allons remédier, ce qui va nous permettre de voir de nouvelles instructions Basic ou de nouvelles formes de celles que nous connaissons. Nous suivrons d'ailleurs cette méthode tout au long du livre.

Objection n° 1 : Lorsque l'ORIC affiche le ? au cours de l'instruction INPUT, rien ne dit que c'est à un rayon qu'il s'attend ; cela n'est pas gênant car nous le savons, mais une personne qui ne connaîtrait pas le programme ne saurait pas quoi répondre au point d'interrogation. Même l'auteur du programme, dans le cas d'un programme qui a besoin de beaucoup de données, peut

avoir besoin qu'on lui remémore dans quel ordre entrer celles-ci.

En résumé, ce qui serait souhaitable, c'est d'afficher un message qui dise à l'utilisateur quelles sont les données attendues par l'INPUT. On aimerait, par exemple, avoir un affichage du genre :

RAYON DU CERCLE ? ou DONNEZ UN RAYON ?

Eh bien Basic le permet très facilement. En effet, on peut employer INPUT sous la forme :

```
INPUT "texte" ; variable
```

A ce moment, le texte entre les guillemets va apparaître sur l'écran, suivi du point d'interrogation habituel de INPUT, et il suffit de répondre comme dans le cas précédent.

Ainsi, si on remplace l'instruction 1Ø du programme A-1 par :

```
1Ø INPUT "RAYON DU CERCLE" ; R
```

notre problème sera complètement résolu. Le point-virgule qui sépare le texte de la liste de variables est impératif, tout comme les guillemets qui délimitent le texte.

Oui, mais comment remplacer l'ancienne instruction 1Ø par la nouvelle ? Tout simplement en tapant la nouvelle instruction 1Ø, sans oublier de commencer par le numéro. Elle viendra remplacer l'ancienne dans la mémoire où le programme est conservé.

Objection n° 2 : Lorsque le résultat est affiché, on obtient un nombre, mais rien n'indique qu'il s'agisse de la surface du cercle. On aimerait bien, là aussi, qu'un message convenable nous éclaire. Dans le cas d'un programme fournissant beaucoup de résultats, il serait tout à fait indispensable que chaque résultat soit identifié.

Comme le précédent, ce problème se résout facilement en Basic. Il suffit de savoir qu'on peut, par PRINT, afficher n'importe quel texte entouré de guillemets. Ainsi, nous pourrons remplacer l'instruction 3Ø par :

```
3Ø PRINT "SURFACE =",S
```

et tout sera dit.

On aura maintenant un dialogue de la forme (nous soulignons ce qui est tapé par l'utilisateur) :

```
RUN
RAYON DU CERCLE ? 1Ø
SURFACE = 314.159265
Ready
```

Objection n° 3 : Notre programme est nettement amélioré, surtout du point de vue du dialogue homme-machine, et toute entrée-sortie doit toujours se présenter ainsi. Mais il reste encore un petit élément d'inconfort : lorsqu'on veut traiter plusieurs rayons, il faut à chaque fois taper RUN, ce qui est un peu fastidieux.

Basic a une réponse : nous allons ajouter à la fin du programme une instruction qui dit à l'ORIC "recommencez l'exécution depuis le début (=l'instruction 1Ø)". Ajoutons l'instruction :

```
4Ø GOTO 1Ø
```

Sachant que "GOTO" signifie "aller à", le fonctionnement est évident : à chaque fois qu'un calcul est fait et son résultat imprimé, l'ORIC arrive à l'instruction 4Ø qui le renvoie en 1Ø où on demande un nouveau rayon et ainsi de suite ...

On dispose maintenant du programme A-2 qui, bien qu'il ait été facile à écrire, a un comportement très commode. Il fait apparaître une notion importante, la notion de **boucle.** Une structure formée d'un groupe d'instructions qui seront exécutées plusieurs fois s'appelle une boucle. La possibilité d'exécuter des boucles, donc d'effectuer des calculs itératifs est un point fort des ordinateurs.

Programme A-2

```
1Ø INPUT "RAYON DU CERCLE";R
2Ø S=PI*R↑2
3Ø PRINT "SURFACE = ",S
4Ø GOTO 1Ø
```

Sortie du programme par 'Ctrl' C.
Commande CONT.

La boucle que nous venons de voir dans le programme A-2 résolvait un problème, mais elle nous en pose un autre : en effet, elle ne finit jamais. Indéfiniment l'ORIC demandera un nouveau rayon, et encore un autre...

Il est en fait normalement interdit d'implanter une telle boucle indéfinie dans un programme ; tout programme doit être assuré de se terminer au bout d'un temps fini. Nous verrons par la suite des instructions permettant d'établir des boucles dont on est sûr qu'elles se terminent. Cependant, on a un moyen de s'en sortir.

Ce moyen est nécessaire car, malheureusement, il peut arriver que les choses s'arrangent mal. Si, par suite d'une erreur de programmation, l'ordinateur se perd dans une boucle sans fin et tourne indéfiniment sans fournir de résultat, ne peut-on pas reprendre le contrôle ?

Si ! Il suffit d'appuyer sur la combinaison de touches 'Ctrl' et C. On obtient aussitôt l'affichage :

```
BREAK IN    (arrêt à l'instruction numéro...)
Ready
```

On peut alors faire imprimer des variables pour voir si tout est correct. Il se peut, en effet, que l'on n'obtienne pas de résultats pendant longtemps, simplement parce que les calculs sont longs et non parce que l'on a une boucle indéfinie. A ce moment, on peut faire reprendre l'exécution là où elle en était, en tapant la commande en mode direct : CONT.

'Ctrl' C fait arrêter l'exécution du programme de façon que celle-ci puisse reprendre par CONT. CONT ne peut fonctionner que s'il n'y a eu aucune modification du programme pendant l'arrêt.

Le programme A-2 est donc correct : nous pouvons avoir la surface de tous les cercles que nous voulons, et quand nous n'en voulons plus, nous frappons les touches 'Ctrl' et C pour terminer.

N.B. Le message BREAK IN... n'apparaît pas lorsque c'est un INPUT qu'on interrompt par 'Ctrl' C.

Compléments : Sans prétendre être complets (le meilleur moyen d'apprendre toutes les particularités d'un langage, c'est la pratique) il y a un ou deux détails supplémentaires que nous devons donner maintenant sur INPUT et sur PRINT.

Entrée de plusieurs données

On peut entrer plusieurs données dans une même instruction INPUT. Par exemple, si au lieu de calculer la surface du cercle, nous voulions calculer le volume d'un

cylindre, il faudrait donner le rayon, mais aussi la hauteur H.

On pourrait employer une instruction de la forme :

10 INPUT "RAYON,HAUTEUR";R,H

et, en réponse, il faudrait maintenant taper deux nombres séparés par une virgule :

RAYON,HAUTEUR ? 15.5 , 20 'Return'

Ne confondez pas le point de 15.5 qui, en notation anglaise sépare partie entière et partie décimale d'un nombre, avec la virgule qui sépare deux nombres différents qui iront dans des variables différentes.

Petit "courrier du coeur" : *En réponse à un INPUT I,J où je devais entrer les valeurs 45 et 105, j'ai, par erreur, mal placé la virgule et tapé 4,5105. Que se passe-t-il ? :* **...** l'ORIC va entreprendre les calculs avec les valeurs I = 4 et J = 5105, qui ne sont pas celles que vous souhaitiez.

Autre question : *à un INPUT I,J où je devais fournir deux valeurs, je n'en n'ai donné qu'une, suivie de 'Return'. Que se passe-t-il ? :* **...** rien de grave, l'ORIC va prendre en compte cette première valeur et, voyant qu'il lui en faut une autre, il affichera un nouveau point d'interrogation. Lorsqu'on a un INPUT à plusieurs variables, on peut grouper les données qu'on fournit comme on veut, puisque, tant qu'il n'aura pas eu toutes les valeurs qu'il attendait, l'ORIC vous le signalera en affichant un point d'interrogation.

Exemple de dialogue : (instruction 10 INPUT"I,J ";I,J)

```
I,J ? 45 'Return'
?   105 'Return'
```

Au contraire, maintenant, je tape trop de valeurs (exemple : 45,105,200 pour INPUT I,J). Que se passe-t-il ? : **...** l'ORIC prend en compte les n premières valeurs s'il en attend n (dans notre exemple I = 45 et J = 105) et il imprime EXTRA IGNORED ce qui indique que les valeurs supplémentaires (et superflues) sont ignorées.

Et si je ne tape pas du tout de valeurs, c'est-à-dire si, en réponse à un INPUT, je fais 'Return' tout de suite ? : **...** l'ORIC affiche un point d'interrogation pour montrer qu'il attend des données.

Expression arithmétique dans un ordre PRINT

Nous avons, jusqu'ici, demandé l'impression soit d'un titre, soit de la valeur d'une variable. Ce sont des cas particuliers de la loi générale qui permet de mettre comme élément à imprimer n'importe quelle expression arithmétique : l'ORIC effectuera le calcul et affichera la valeur obtenue.

Tapez

```
?"5 A POUR CARRE",5↑2
```

Vous obtiendrez

```
5 A POUR CARRE 25
```

Comme cas particulier d'expression arithmétique, on peut mettre une variable, mais même une constante. On pourrait écrire :

```
? 5 ; "A POUR CARRE", 5↑2
```

à la place de l'exemple précédent.

Lorsqu'on fait imprimer un titre, on exploite ce fait : tout texte encadré de guillemets constitue une constante chaîne de caractères.

Séparation des éléments dans une liste d'impression

Il est bien entendu que là où nous disons "impression" il faut entendre "affichage sur l'écran", mais avec l'ORIC il suffit d'une seule modification que nous verrons bien plus tard pour que toutes les informations concernées par tous les ordres d'impression apparaissent à l'imprimante si l'on en dispose d'une.

Finissons-en aussi avec le point d'interrogation : sur l'ORIC, ? est une abréviation commode de l'instruction PRINT.

Par ailleurs, dans les différents exemples qui ont précédé, lorsque nous avons voulu afficher plusieurs informations dans le même ordre PRINT, nous les avons séparées tantôt par point-virgule, tantôt par virgule. Quelle est la différence ? Lorsque deux zones sont séparées par point-virgule, elles seront imprimées jointives sur la ligne, tandis que si on les sépare par une virgule, l'impression de la seconde zone commencera à la colonne 6 (on appelle cela la tabulation ; en fait, elle consiste à laisser 3 espaces).

Essayez :

```
?A,B
Ø     Ø
?A;B
Ø Ø
```

Pourquoi les zéros ne sont-ils pas "collés" dans le second exemple ? Parce que chaque nombre imprimé comporte un signe (ici on n'a rien car les nombres sont positifs) et est suivi d'un espacement.

Qu'en est-il pour les chaînes de caractères ?

Essayez :

```
?"BON";"JOUR"
BONJOUR
?"BON","JOUR"
BON   JOUR
?"EH  BONJOUR","MONSIEUR"
EH  BONJOUR   MONSIEUR
```

Encore deux indications :

- Si nous ne mettions aucun séparateur, l'ORIC ferait comme si nous avions mis un point-virgule. Mais cela n'est pas possible qu'entre deux chaînes de caractères, ou entre chaîne et variable.

- On peut terminer l'ordre PRINT par une , ou un ; alors qu'il n'y a rien à séparer. Cela a pour effet que le prochain ordre PRINT écrira sur la même ligne, de façon jointive ou avec une tabulation, selon que l'on aura mis ; ou ,

Essayez les programmes :

```
1              2               3

1Ø ?"BON"      1Ø ?"BON";      1Ø ?"BON",
2Ø ?"JOUR"     2Ø ?"JOUR"      2Ø ?"JOUR"
```

Effet

```
BON            BONJOUR         BON      JOUR
JOUR
```

Tout vient de ce qu'un ordre PRINT normal effectue un "retour chariot" pour terminer, qui est supprimé par le ; ou la ,
Par contre, si l'on veut avancer d'une ligne sans rien imprimer, il suffit de mettre PRINT tout court.

Précisons enfin que, si pour un PRINT la , ou le ; ne font pas grande différence sauf si l'on veut soigner la mise en page, pour INPUT ils sont essentiels et pas du tout interchangeables : les variables à entrer doivent être séparées par des virgules (ainsi que les données au moment où on les fournit) et, s'il y a un message de titre, il doit être séparé du reste par un point-virgule.

EXERCICES

Bien que nous n'ayons vu que trois instructions Basic, nous avons déjà en main des possibilités immenses, car elles couvrent les trois opérations fondamentales de tout traitement :

- entrer les données,
- calculer,
- sortir les résultats.

Vérifions-le sur deux exercices que nous vous recommandons instamment de traiter sans regarder la solution à la fin du volume.

Exercice 2.1

Modifier le programme A-2 pour calculer le volume de cylindres de rayon R et hauteur H.

Exercice 2.2

Ecrire un programme de structure analogue, mais qui calcule l'intérêt rapporté par un certain capital placé à un certain taux pendant N années (intérêts composés annuellement).

Remarque : la numérotation des exercices est faite sous la forme chapitre.numéro.

CHAPITRE III

COMMANDES FONDAMENTALES

Un programme entré en mode différé ne peut être utilisé qu'en conjonction avec un certain nombre de commandes en mode direct, qui disent au système d'exploitation ce que l'on veut faire.

Nous avons déjà vu la plus fondamentale de ces commandes : RUN qui permet d'exécuter le programme.

Mais on peut faire d'autres opérations sur un programme. On peut le lister, c'est-à-dire imprimer ses instructions, ce qui est utile, notamment pour rechercher des erreurs, ou si l'on envisage une modification. On peut le corriger et, pour cela, l'ORIC est assez commode. On peut le sauvegarder sur cassette pour pouvoir l'utiliser ultérieurement sans avoir à le retaper.

Le Basic de l'ORIC possède tout un environnement de commandes permettant ces opérations. Il est indispensable que nous les voyions maintenant.

LIST

LIST tout court fournit sur l'écran la copie de tout le programme présent en mémoire. S'il n'y a pas de programme, par exemple, aussitôt après la mise sous tension, l'ORIC affiche Ready immédiatement. On dit qu'on obtient la liste, ou en franglais le listing du programme.

Si le programme est très long, et donc ne tient pas dans les 28 lignes de l'écran, la liste va défiler sur l'écran (une ligne apparaît en bas alors qu'une ligne disparaît en haut) et la lecture sera difficile. Pour la faciliter, on peut stopper en appuyant sur la barre d'espace. Le listing reprend en appuyant sur n'importe quelle touche.

Vous pouvez l'essayer, mais les programmes que nous avons écrits jusqu'à présent sont un peu courts pour que cela se voie bien.

Deux particularités sont apparues sur les listes que nous avons pu obtenir en faisant quelques essais.

- les instructions apparaissent sur la liste dans l'ordre des numéros croissants, c'est-à-dire le même ordre que pour l'exécution, quel que soit l'ordre dans lequel elles ont été tapées ;
- si, pour des instructions d'impression, nous avons utilisé le point d'interrogation, sur la liste c'est le mot PRINT qui apparaît.

Nous supposerons dans la suite que nous avons en mémoire, le programme A-2 du chapitre précédent.

Listes partielles

On peut lister une seule instruction, en donnant son numéro : *LIST x*

```
LIST 2Ø
2Ø S=PI*R↑2
```

Pour lister toutes les instructions comprises entre les instructions de numéros x et y, on tape *LIST x - y*

```
LIST 2Ø - 3Ø
2Ø S=PI*R↑2
3Ø PRINT "SURFACE = ",S
```

Les bornes - si elles existent - sont comprises. LIST 15 - 35 donnerait le même résultat que l'exemple précédent.

LIST -x : liste depuis le début jusqu'à la ligne x :

```
LIST -2Ø
1Ø INPUT "RAYON DU CERCLE";R
2Ø S=PI*R↑2
```

LIST x- : liste à partir de la ligne x jusqu'à la fin :

```
LIST 3Ø-
3Ø PRINT "SURFACE = ",S
4Ø GOTO 1Ø
```

Comme l'exécution d'un programme, une liste peut être interrompue à l'aide des touches 'Ctrl' C. Couplé avec le LIST x-, cela permet de lister un long programme par morceaux : on appuie sur 'Ctrl' C quand on voit que l'écran va être plein.

Bien sûr, s'il n'y a aucune instruction dans l'intervalle demandé, on a tout de suite Ready.

NEW

Lorsqu'on tape une instruction Basic en mode programmé (donc avec un numéro) il peut se passer deux choses :

- ou bien l'instruction que l'on vient de taper porte le même numéro qu'une instruction déjà présente : à ce moment elle vient *remplacer* l'ancienne ;
- ou bien il n'existait pas d'instruction de même numéro que celle qu'on vient de taper : alors - comme on pourrait le constater en demandant LIST - l'instruction vient *s'intercaler* dans le programme à la place impliquée par son numéro.

Il en résulte que, si l'on veut introduire un programme complètement nouveau, et si les instructions nouvelles ne correspondent pas une à une à celles de l'ancien programme, il restera des instructions anciennes au milieu des nouvelles, qui, bien entendu, perturberont le fonctionnement.

La commande *NEW* a pour but d'éliminer cet inconvénient : son effet est de supprimer complètement le programme actuellement présent. Il est conseillé de l'utiliser avant de taper un nouveau programme.

NEW ne doit pas être confondue avec une autre commande ou instruction *CLEAR* qui, elle, a pour effet de remettre à zéro toutes les variables. En somme, NEW vide la mémoire programme tandis que CLEAR vide la mémoire des données (l'une et l'autre sont deux zones de la même mémoire).

Enfin, une autre instruction, *CLS*, vide l'écran.

Remarque : En fait, NEW contient CLEAR, c'est-à-dire que lorsqu'on fait NEW, toutes les variables sont, par la même occasion, remises à zéro. De même, RUN contient CLEAR, ce qui faut qu'au début de l'exécution d'un programme, toutes les variables ont la valeur 0 jusqu'à ce qu'une instruction leur donne une autre valeur.

RUN

Nous connaissons déjà bien la commande RUN tout court. On peut l'employer aussi sous la forme *RUN x* où x est un numéro d'instruction. Cela aura pour effet de lancer l'exécution du programme, mais à partir de la ligne x.

Question : *J'ai un programme comportant l'instruction 1Ø. Je peux le lancer par RUN 1Ø . Je peux aussi le lancer en tapant GOTO 1Ø en mode direct. Quelle est la différence ?*
- RUN 1Ø remet les variables à zéro, ce que ne fait pas GOTO 1Ø. Introduisez comme programme :

```
5 A = 3
1Ø ?A
```

et essayez les dialogues

1	**2**	**3**
A=5	A=5	A=5
RUN	RUN 1Ø	GOTO 1Ø

Effet

3	Ø	5

Dans le premier cas, on passe sur l'instruction 5 qui donne à A la valeur 3. Dans le second cas, RUN remet A à zéro. Ce n'est que dans le troisième cas que l'effet de l'affectation en mode immédiat sera conservé.

END

Le RUN numéro permet de constituer un programme en plusieurs parties telles que l'on exécute tantôt l'une tantôt l'autre. Il suffit de taper RUN numéro de la première instruction de la partie voulue.

Oui, mais supposons que l'on ait exécuté la première partie : on va tomber maintenant sur la deuxième partie, ce qui n'est peut être pas souhaité. Il suffit de

terminer chaque partie par une instruction *END* qui veut dire "retourner au niveau de commande directe". Après exécution d'une instruction END, Basic affiche Ready. Pour la bonne règle, notre programme A-1 aurait dû se terminer par une instruction :

```
40 END
```

Mais, en fait, lorsque Basic arrive à la dernière instruction d'un programme sans rencontrer de END, il fait comme s'il y avait cette instruction.

Edition d'un programme

Nous devons voir maintenant tout un ensemble de procédures qui permettent de modifier ou corriger un programme en ayant le moins possible à retaper.

De ce point de vue, l'ORIC est assez commode. Mais avant de voir ces procédures, il nous faut faire plus ample connaissance avec le clavier. Nous aurions pu le faire précédemment, avant même de procéder à nos premiers essais, mais nous avons pu nous en passer, alors que maintenant c'est indispensable.

Le clavier de l'ORIC

Le clavier de l'ORIC est représenté *figure 3-1*.

Les touches de l'ATMOS sont de taille et de toucher plus professionnels, mais leur disposition est la même.



Figure 3-1 : le clavier de l'ORIC

On peut considérer qu'il y a trois sortes de touches :

- les touches ordinaires, dont l'appui fait afficher le caractère correspondant sur l'écran : par exemple, quand vous appuyez sur la touche A, il s'imprime un A sur l'écran ;

- les touches de modification de l'affichage et les touches spéciales : comme 'Return' ou les touches de mouvement de curseur ;
- les touches de commande qui, enfoncées simultanément avec une seconde touche, déterminent la fonction de cette seconde touche. Les touches de commande sont 'SHIFT' (il en existe deux qui sont parfaitement équivalentes), 'Ctrl' (Contrôle) et 'Esc' (Escape).

Chaque touche ordinaire a, en général, deux fonctions, représentées l'une au dessus de l'autre, au dessus de la touche. (Ex. ?/). Vous obtenez le caractère du bas (Ex. /) en appuyant simplement sur la touche. Vous obtenez le caractère du haut (Ex. ?) en appuyant en même temps sur SHIFT.

Lorsque nous disons "en appuyant simultanément", cela signifie : appuyer d'abord sur la touche de commande (SHIFT ou CTRL) et la maintenir enfoncée, appuyer sur la touche voulue, la relâcher, et, enfin, relâcher la touche de commande. Entraînez-vous à votre clavier !

Le signe ↑ (élévation à la puissance) est marqué ^ sur le clavier ('Shift' 6).

Pour les touches lettre, il y a deux modes : majuscules seules (valable à la mise sous tension) et majuscules/minuscules où, sans 'Shift' on a la minuscule, et avec 'Shift' on a la majuscule. On passe d'un mode à l'autre en faisant 'Ctrl' T. Les programmes Basic doivent être tapés en majuscules.

Touches mouvement de curseur

Le curseur est le rectangle clignotant que vous avez à l'affichage lorsque l'ORIC attend que vous tapiez quelque chose : il marque la position sur l'écran où apparaîtra le prochain caractère que vous taperez.

Les touches mouvements de curseur déplacent ce curseur sans imprimer de caractère ni modifier les caractères déjà présents à l'écran, sur lesquels le curseur passe.

Les touches →, ↑ , ← , et ↓ créent , de façon évidente le mouvement marqué.

Exemple : ↑ que nous figurerons sous la forme h fait aller d'un cran vers le haut. De même, nous utiliserons

les abréviations suivantes pour les symboles flèches en bas b, flèche à gauche g, flèche à droite d.

Exercice d'entraînement 3.1

Effectuez le parcours figure 3.2 en prenant bien soin de revenir au point A.

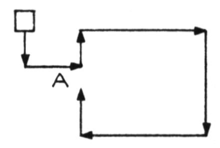


Figure 3-2

La touche 'Del' supprime le caractère immédiatement à sa gauche, permettant ainsi une correction immédiate des erreurs de frappe.

Nous voulons taper BONJOUR, mais nous nous apercevons que nous avons tapé BONKO
Nous tapons une première fois sur 'Del' : BONK
puis une deuxième : BON et nous n'avons plus qu'à continuer JOUR.

Exercice d'entraînement 3.2

Vous vouliez taper BONJOUR et vous avez tapé BONUR

Ces dernières touches vont spécialement nous servir pour la correction des programmes.

La touche 'Ctrl'

La touche 'Ctrl' produit des fonctions spéciales lorsqu'elle est appuyée en même temps que certaines autres touches.

Actions en va et vient

Un appui produit une action, un deuxième appui produit le contraire.

'Ctrl' D   passe en mode doubles caractères en hauteur
'Ctrl' F   active/supprime le bip à chaque appui de touche
'Ctrl' P   active l'imprimante
'Ctrl' Q   active/supprime le curseur
'Ctrl' S   active l'écran

'Ctrl' T — passe en mode majuscules seules.
'Ctrl'] — permet d'accéder aux deux colonnes de gauche de l'écran (la ligne passe ainsi à 40 colonnes).

Formatage sur l'écran

'Ctrl' H — curseur à gauche.
'Ctrl' I — curseur à droite.
'Ctrl' J — curseur bas.
'Ctrl' K — curseur haut.
'Ctrl' L — vidage écran.
'Ctrl' M — retour-chariot.
'Ctrl' N — vide la ligne où se trouve le curseur.

Actions spéciales

'Ctrl' A — se comporte comme un curseur à droite mais tout se passe comme si les caractères sur lesquels on passe venaient d'être retapés. C'est un des outils essentiels de la correction des programmes.
'Ctrl' C — stoppe l'exécution d'un programme ou d'un listing.
'Ctrl' G — fait retentir un "ding".
'Ctrl' X — arrête la frappe d'une ligne.

La touche 'Esc' (escape)

Attention, à la différence de 'Ctrl' et 'Shift', vous appuyez sur '**Esc**' puis sur le second caractère.

Cette touche a pour effet de modifier les attributs de l'affichage écran, à partir du moment où elle est tapée jusqu'à la fin de la ligne ou jusqu'à la prochaine modification. Les attributs seront traités plus en détail dans un autre chapitre. Disons ici qu'ils déterminent la couleur du fond et la couleur des caractères, le fait d'être en simple hauteur ou en double, le fait d'être fixe ou clignotant, le fait d'être pris dans le jeu de caractères standard ou auxiliaire.

'Esc' suivi de	*produit*	*'Esc' suivi de*	*produit*
@	caractères noirs	N	double hauteur, clign, std
A	caractères rouges	O	double hauteur, clign, aux
B	caractères verts	P	fond noir
C	caractères jaunes	Q	fond rouge
D	caractères bleus	R	fond vert
E	caractères pourpres	S	fond jaune
F	caractères turquoises	T	fond bleu
G	caractères blancs	U	fond pourpre
H	simple hauteur, fixe, std	V	fond turquoise
I	simple hauteur, fixe, aux	W	fond blanc
J	double hauteur, fixe, std	X	texte 60 Hz
K	double hauteur, fixe, aux	Y	texte 60 Hz
L	simple hauteur, clign, std	Z	texte 50 Hz
M	simple hauteur, clign, aux	{	texte 50 Hz
		}	graphique 60 Hz
		del	graphique 50 Hz

Notes : *std = jeu de caractères standard*
aux = jeu de caractères auxiliaire
clign = clignotant.

le 50 Hz est applicable en Europe, le 60 Hz aux U.S.A.

Si, en Europe, vous faites un 'Esc' X, vous perdez la synchronisation de l'affichage.

Exercice 3.3

Ecrivez (en mode direct) BONJOUR MADAME BONJOUR MONSIEUR avec les deux BONJOUR fixes sur fond blanc, le premier noir, le deuxième bleu. MADAME clignotant blanc sur fond noir, MONSIEUR clignotant jaune sur fond bleu.

Autres touches spéciales

Nous avons déjà vu les rôles de 'Return' (retour-chariot et acquittement de message) et 'Barre d'espace' (espace et arrêt temporaire d'un listing)

Mode minuscule

A la mise sous tension, les touches lettres donnent la lettre majuscule, avec Shift ou sans Shift.

Eh bien il existe un mode où l'on obtient la lettre minuscule (avec la touche seule), la lettre majuscule (avec Shift). On a donc un comportement de machine à écrire.

On passe d'un mode à l'autre, et inversement, en appuyant simultanément sur 'Ctrl' et T. La transformation affecte tout ce qui est affiché sur l'écran.

Répétition

Toutes les touches ont la répétition automatique, c'est-à-dire que si on les maintient appuyées, leur fonction se répète. C'est spécialement pratique pour faire voyager rapidement le curseur.

L'ORIC ATMOS a exactement la même disposition de touches sauf qu'il a en plus, en bas à droite, une touche FUNCT qui n'a pas reçu de rôle spécial mais dont l'appui peut être décelé (cf p. 104) de la même façon que SHIFT ou CTRL.

Modification ou correction d'un programme

Lorsqu'on a trouvé une erreur dans un programme, ou lorsqu'on veut simplement y apporter une modification, il est important de pouvoir le faire commodément, c'est-à-dire en ayant le moins possible d'informations à retaper.

Nous savons déjà :

- *Ajouter ou insérer une nouvelle instruction* : il suffit de taper l'instruction en lui attribuant un numéro compris entre ceux des instructions entre lesquelles on veut l'intercaler.

Il en résulte immédiatement un conseil : lorsqu'on écrit la première version d'un progamme, il ne faut pas donner des numéros consécutifs, afin de pouvoir faire des insertions par la suite. On suggère, par exemple, de numéroter de 10 en 10 ;

- *Transformer complètement une instruction* : on tape la nouvelle version avec le même numéro que l'ancienne. Dès qu'on tape 'Return' le remplacement s'effectue.

Voyons maintenant de nouvelles possibilités :

- *Suppression complète d'une instruction* : il suffit de taper le numéro suivi de 'Return'.

Nous allons en faire un essai qui va nous servir pour la suite. Nous supposons que nous avons le programme A-1 en mémoire (sinon, retapez-le : c'est fastidieux, mais nous verrons bientôt comment récupérer un programme sans le retaper, grâce aux cassettes).

Faisons un LIST, mais, avant, nous tapons CLS pour avoir le listing en haut de l'écran.

Ensuite, nous tapons 3Ø 'Return' pour supprimer la dernière instruction. Nous vérifions par LIST que cela a bien été fait. Nous n'avons plus l'instruction 3Ø.

Pour la récupérer, si par suite d'une erreur ce n'était pas celle-là qu'il fallait supprimer, on doit théoriquement la retaper. Eh bien non ! Pas tant que l'instruction est affichée. En effet, tant que l'instruction est affichée, elle est dans la mémoire d'écran. Il y a un moyen de la transférer dans la mémoire programme : c'est de ramener le curseur au début de la ligne de l'instruction et de parcourir la ligne en faisant 'Ctrl' A ; on termine par 'Return'

Essayons-le : nous amenons le curseur sur la ligne 3Ø du premier listing ; on a, à l'affichage :

▨ Ø PRINT S

On fait des 'Ctrl' A jusqu'à 3Ø PRINT S ▨

A ce moment, on tape 'Return'. Il semble ne rien se passer. Descendons le curseur vers le bas de l'écran et faisons LIST : on voit alors que l'instruction 3Ø est revenue dans le programme.

En résumé, dès qu'on tape 'Ctrl' A en face d'un caractère sur l'écran, c'est comme si on le retapait. C'est ce qui va nous servir pour toutes les autres procédures de modification. C'est déjà ce qui est exploité pour supprimer une instruction : on crée une nouvelle version vide de l'instruction, et Basic ne garde pas une instruction vide.

- *Modifications de quelques caractères sur une ligne* : C'est là que se manifeste toute la puissance du système d'édition de l'ORIC. Il résulte de ce que nous venons de voir que la procédure à observer est la suivante :

1 - Lister la ligne à modifier (si elle n'est pas déjà sur l'écran).

2 - Amener le curseur sur la ligne à modifier et aller par 'Ctrl' A jusqu'au premier caractère à changer.

3 - Pour chaque caractère à remplacer, taper sur place le nouveau caractère. S'il y a des caractères à supprimer, les passer par mouvement de curseur ; s'il y en a à insérer, aller (par curseur) à une ligne vide de l'écran et taper les caractères voulus.

4 - Revenir à la ligne concernée par curseur pour copier par 'Ctrl' A les caractères à garder . Terminer par 'Return'.

On peut modifier une ligne en autant d'étapes qu'on veut : il suffit de revenir sur la ligne, y faire d'autres modifications après avoir tapé 'Return'. Il est conseillé de vérifier par LIST que les modifications souhaitées ont bien été faites.

- *Modification du numéro d'une ligne* : C'est un cas particulier du précédent : c'est maintenant le numéro qu'on change, mais attention ! Une fois qu'on a tapé 'Return' il y a deux lignes identiques, une à l'ancien numéro et une autre au nouveau. Cela peut être spécialement précieux pour économiser le temps de frappe lorsque l'on a toute une série de lignes très voisines à taper.

Supposons que l'on veuille avoir (c'est un cas d'école) :

```
50 GOTO 10
70 GOTO 10
90 GOTO 10
95 GOTO 12
```

On tapera :

```
50 GOTO 10 'Return'
```

puis, curseur sur le 5 :

7 des 'Ctrl' A 'Return'

puis, curseur sur le 7 :

9 des 'Ctrl' A 'Return'

puis, curseur sur le 9 :

95 des 'Ctrl' A jusqu'au Ø 2 'Return'

A chaque fois, on n'aura sur l'écran que la dernière ligne tapée, mais en faisant LIST on s'aperçoit que l'on dispose de toutes les lignes voulues.

Exercice 3.4

Modifier le programme A-1 pour qu'il calcule, non pas la surface du cercle, mais le volume de la sphère de rayon R.

Nous donnons l'exercice moins pour le calcul que pour s'entraîner à effectuer la modification. On voit facilement qu'il suffit de changer le nom du résultat en V (il serait possible de garder S, mais nous voulons des identificateurs parlants). Donc, 30 doit devenir 30 PRINT V, tandis que 20 doit devenir 20 V=4/3*PI*R↑3.

La modification de 30 est évidente, on amène le curseur sur le S par 'Ctrl' A et on tape V 'Return'.

Attention : tout caractère non copié par 'Ctrl' A est oublié.

Pour l'instruction 20, on amène par 'Ctrl' A le curseur sur le S qu'on change en V, puis, curseur sur le P, on va à un endroit vide de l'écran taper 4/3*. On revient sur le P par curseur et on termine la ligne par 'Ctrl' A jusqu'au 2 sur lequel on tape 3.

***Question** : Ne manque-t-il pas quelque chose ? :* si ! le 'Return'.

Bien sûr, il aurait mieux valu modifier le programme A-2 ; c'est l'objet de l'exercice 3.5.

Exercice 3.5

Faire la même modification sur le programme A-2.

Tout devrait bien se passer.

Exercice d'entraînement 3.6

Faire le passage du programme A-2 au calcul du volume du cylindre.
(Exercice 2.1, solution en annexe).

Avant de voir les commandes qui concernent les cassettes, nous passons à un autre exercice qui va nous permettre d'aborder une autre sorte de problèmes.

Exercice 3.7

Ecrire un programme analogue au programme A-2, mais qui, cette fois, demande la surface d'un cercle et en déduise le rayon.

Le problème est que, cette fois, nous n'avons pas une formule connue "toute cuite" à appliquer. Il faut la chercher, encore que, pour cet exercice, ce ne soit pas trop sorcier.

Néanmoins, et ce sera vrai dans tous les problèmes autres que ceux qui sont totalement évidents, l'ordinateur n'est pas capable de trouver tout seul la solution d'un problème. Il faut la lui donner sous forme d'une suite ordonnée d'opérations à effectuer. Une telle suite, qui n'est rien d'autre qu'une recette, s'appelle chez les informaticiens savants un algorithme.

Il est presque toujours plus difficile de trouver l'algorithme résolvant un problème que de programmer cet algorithme une fois qu'on l'a trouvé.

Revenons à notre exercice. On trouve facilement que la formule à appliquer est $R = \sqrt{S/PI}$. Comment allons-nous traduire la racine carrée ? Eh bien nous avons le choix, ce qui arrive souvent en programmation, entre ↑.5 (puissance 1/2), ou appeler la fonction méthématique SQR (racine carrée) qui est une de celles dont on dispose en Basic. D'où deux solutions pour l'instruction 2Ø du programme A-3.

Programme A-3

```
10 INPUT "SURFACE DU CERCLE";S
20 R = SQR(S/PI)
30 PRINT "RAYON = ",R
40 GOTO 10
```

Autre forme de 2Ø :

```
20 R = (S/PI)↑.5
```

Rangement d'un programme sur cassette

(Si vous disposez d'un magnétophone et avez fait le branchement décrit dans la notice, ce qui est vivement recommandé).

Voici maintenant des commandes spécialement utiles. En effet, jusqu'ici nous n'avons écrit que des programmes très courts et peu nombreux. Néanmoins, même ainsi,

il est fastidieux de les taper plusieurs fois, et cela entraîne des risques d'erreurs. Or si nous faisons NEW, ou si nous éteignons l'ORIC simplement pour aller nous coucher, le programme est perdu (la mémoire vive RAM perd ses informations lorsqu'elle n'est plus alimentée). Heureusement, il y a les cassettes sur lesquelles on peut sauver des programmes et les relire par la suite.

Sauvegarde

Munissez-vous d'une cassette vierge rebobinée (sinon, vous la rebobinez avec la touche REW du magnétophone). Notons, à ce propos, qu'il est recommandé d'utiliser des cassettes courtes (il vaut mieux, étant donné les temps de lecture, ne ranger que peu de programmes sur chaque cassette) et de bonne qualité.

Supposons que nous ayons un programme précieux en mémoire, par exemple le programme A-3 que nous venons de faire. Placez la cassette dans le magnétophone et tapez :

```
CSAVE "RAYON-CERCLE"
```

Si votre magnétophone n'a pas de télécommande, il faut préalablement appuyer sur les touches RECORD et PLAY (simultanément). S'il a une télécommande convenablement connectée, vous ne vous souciez de rien et l'ORIC affiche :

```
SAVING RAYON-CERCLE
```

Lorsque Ready et le curseur réapparaissent, c'est fini : le programme a été écrit sur la cassette sous le nom que nous avons donné.

Remarque : Le nom peut comporter jusqu'à 17 caractères. Il peut contenir des points ou des tirets.

Chargement

Pour renvoyer en mémoire un programme préalablement sauvegardé sur une cassette, on utilise la commande CLOAD "RAYON-CERCLE" (après avoir mis le magnéto en lecture).

L'ORIC répond :

```
      Searching      (recherche)
puis  Loading        (lorsqu'il a trouvé le programme
                      cherché)
enfin Ready
```

On peut alors lister le programme ou l'exécuter par RUN.

Si l'on obtient le diagnostic ?FILE ERROR-LOAD ABORTED, il faut rebobiner et réessayer, puis refaire l'essai avec une autre cassette sur laquelle on aura pris la précaution de faire une seconde sauvegarde. En cas d'échec : il faut employer la version lente des commandes magnéto

CSAVE "nom",S   et   CLOAD "nom",S

Sur ORIC ATMOS, on a souvent des messages Errors Found (erreurs rencontrées en lecture) alors que le chargement s'est tout de même bien passé.

Noms abrégés

On peut ne pas spécifier de nom dans la commande CLOAD. Par exemple, on aurait pu écrire :

CLOAD ""   ou   CLOAD "",S

L'ORIC chargera le premier programme rencontré sur la bande.

Chargement et exécution réunis

Si la commande CSAVE a été de la forme

CSAVE "nom",S,AUTO ou CSAVE "nom",AUTO

le programme est exécuté automatiquement (c'est-à-dire sans qu'il y ait besoin de faire RUN) après son chargement. Le "AUTO" n'a pas à figurer dans la commande CLOAD.

Vérification

Après une sauvegarde, il est recommandé de rebobiner la cassette et de faire :

CLOAD "nom", V [,S]

A ce moment, la machine lit le programme sur la cas-
ette et le compare avec la mémoire. Si tout s'est bien
ıssé, on a l'affichage Ø Verify Errors Found. Sinon,
us pouvez recommencer la sauvegarde, éventuellement
mode S.

RECAPITULATION

Nous avons maintenant vu les instructions les plus fondamentales des programmes :

arithmétique
INPUT - PRINT
GOTO - END

Nous avons vu les commandes les plus utiles :

RUN et LIST
CSAVE et CLOAD

ainsi que les procédures de correction des programmes.

Si l'énoncé de certains des mots-clés précédents n'éveille aucun écho en vous, nous vous conseillons, avant de poursuivre, de relire les pages qui les concernent.

Nous sommes maintenant prêts à aborder la seconde série de programmes qui va nous permettre - toujours par variantes successives - d'augmenter notre "arsenal" d'instructions Basic.


TEZ
A
DÉC
RTE
DE
L'ORIC

CHAPITRE IV

BASES DE LA PROGRAMMATION

Si vous voulez bien, nous allons jouer à un jeu. Les programmes de jeu forment une classe importante parmi les programmes pour P.S.I. Tous ne sont pas débiles : certains sont très élaborés et souvent très amusants. D'autre part, la présentation sous forme de jeu de certains programmes pédagogiques les rend plus attrayants donc plus efficaces. Nous souhaitons qu'à la fin de la lecture de ce livre, vous soyez capables, vous aussi, d'en écrire.

Pour le moment, notre jeu sera au départ un peu simple, mais il s'améliorera progressivement. Il s'agit du jeu "devinez un nombre". Le programme connaît un nombre (fixe) et il lit la devinette du joueur ; si le joueur a bien deviné, il affiche "gagné", sinon il affiche "perdu".

L'INSTRUCTION IF

Pour réaliser ce qui est demandé, de quoi avons-nous besoin ? D'une instruction capable de tester une condition (qui sera ici nombre donné par le joueur = nombre caché), c'est-à-dire apprécier si elle est vraie ou fausse ; si elle est vraie, d'aller à une certaine partie du programme (ici, afficher "gagné") sinon d'aller à un autre endroit du programme (ici afficher "perdu").

Eh bien cette instruction existe, c'est l'instruction IF. Sa forme principale est :

```
n IF condition THEN instruction
n'.......
```

Le comportement est le suivant : si la condition est vraie, on effectue l'instruction qui suit THEN, puis l'instruction de la ligne suivante n' ; si la condition est fausse, on passe directement à l'instruction de la ligne n'.

Un cas particulier est celui où l'instruction qui suit THEN est un GOTO :

```
n IF condition THEN GOTO n''
n'
...
n''
```

A ce moment, si la condition est vraie, on va en n'' ; si la condition est fausse, on va en n'.

En fait, il suffit d'écrire l'un des deux mots THEN ou GOTO.

Exemple : 1Ø IF A = B THEN 5Ø
ou 2Ø IF A+B<C GOTO 1ØØ

Nous sommes prêts à écrire notre première version du programme "devinez un nombre".

Programme B-1

```
2Ø INPUT"DEVINEZ UN NOMBRE";A
3Ø IF A=3.25 GOTO 6Ø
4Ø PRINT"PERDU!"
5Ø END
6Ø PRINT"GAGNE"
```

Les instructions INPUT et PRINT sont déjà familières. L'instruction END fait terminer le programme une fois qu'on a inscrit PERDU ! Il n'y en a pas besoin après 6Ø puisque c'est la dernière instruction du programme.

L'instruction IF que nous employons est du troisième type : IF...GOTO. Le nombre à deviner est 3.25 ; le nombre proposé lu au clavier est A ; la condition à tester est "Est-ce que A est égal à 3.25 ?" Eh bien, cela s'écrit A = 3.25. C'est simple.

Du point de vue du fonctionnement du jeu, il y a beaucoup d'objections à formuler sur le programme B-1. Nous

le ferons bientôt, et cela nous aidera à découvrir de nouvelles instructions Basic. Mais la forme présente nous a permis d'utiliser l'instruction IF, qui est une des plus importantes de Basic, et nous avons encore quelques éléments à voir sur cette instruction, en particulier les différentes conditions qui peuvent suivre le IF.

La première forme de condition est : expression arithmétique relation expression arithmétique, comme 2*A + 4<B↑3.

Chacune des expressions arithmétiques est évaluée avant l'examen de la relation. Les opérateurs de relation utilisables sont :

=	égal	< >	différent
<	inférieur	< =	inférieur ou égal
>	supérieur	> =	supérieur ou égal

"différent" s'écrit inférieur ou supérieur, ce qui ne manque pas de logique.

La deuxième forme de condition est une combinaison de relations de la première forme à l'aide des opérateurs logiques AND (et), OR (ou), et NOT (non).

cl AND c2 est vraie seulement si les conditions cl et c2 sont toutes les deux vraies ;

cl OR c2 est vraie dès qu'une au moins des conditions cl ou c2 est vraie ;

NOT c est vraie si c est fausse, et fausse si c est vraie.

- aller en 1ØØ si à la fois C est supérieur à 2*A+4 et B est inférieur à 3 :

```
IF C>2*A+4 AND B<3 GOTO 1ØØ
```

- imprimer OUI si X est extérieur à l'intervalle [1,2[c'est-à-dire X<1 ou X>=2) :

```
IF X<1 OR X>=2 THEN PRINT"OUI"
```

Exercice 4.1

Reprenez le programme A-3. Essayez de fournir une surface négative.

On obtient le message :

? ILLEGAL QUANTITY ERROR IN 2Ø

ce qui est normal, puisque l'on cherche à calculer la racine carrée d'un nombre négatif : il n'y a pas de cercle de surface négative.

Un programme bien écrit doit se garantir contre de telles erreurs de l'opérateur : par exemple, un programme de jeu d'échecs doit vérifier que le coup proposé par le joueur est légal.

L'objet de l'exercice est d'installer une telle garantie dans le programme A-3 : ajoutez au programme une instruction qui renvoie en 1Ø demander une autre surface tant que la surface fournie n'est pas positive. On peut également, en outre, imprimer un message de protestation.

IF... THEN... ELSE

Lorsque le test a la forme d'une *alternative* (***si*** condition ***alors*** faire ceci ***sinon*** faire cela) l'ORIC permet d'éviter les GOTO que donnerait la traduction

```
      IF condition GOTO n
      cela : GOTO p
    n ceci
    p suite.
```

On peut en effet écrire :

```
IF condition THEN ceci ELSE cela
```

"ceci" et "cela" doivent être des instructions Basic (ou cf p 73, des suites d'instructions séparées par des 'deux-points')

Exemple : IF A<B THEN PRINT "A<B" ELSE PRINT "A>B"

Le seul impératif est que tout l'ensemble forme une seule ligne Basic. Après l'exécution de la ligne, on se retrouve à la ligne suivante après avoir fait l'une des deux branches de l'alternative. Ainsi, la séquence

```
1Ø IF A<B THEN PRINT "A" ELSE PRINT "B"
2Ø PRINT "C"
```

fait imprimer soit A soit B
C C

Ce n'est que si l'une des branches contient un GOTO qu'on ne se retrouve pas à la ligne suivante.

Exemple : Calculez Z = valeur absolue de X de trois façons différentes :

1	2	3
5Ø Z=ABS(X)	5Ø Z=X 6Ø IF X<Ø THEN Z=-X	5Ø IF X<Ø THEN Z= -X ELSE Z=X

Bien sûr, notre programme B-1 pourrait s'écrire :

PROGRAMME B-1B

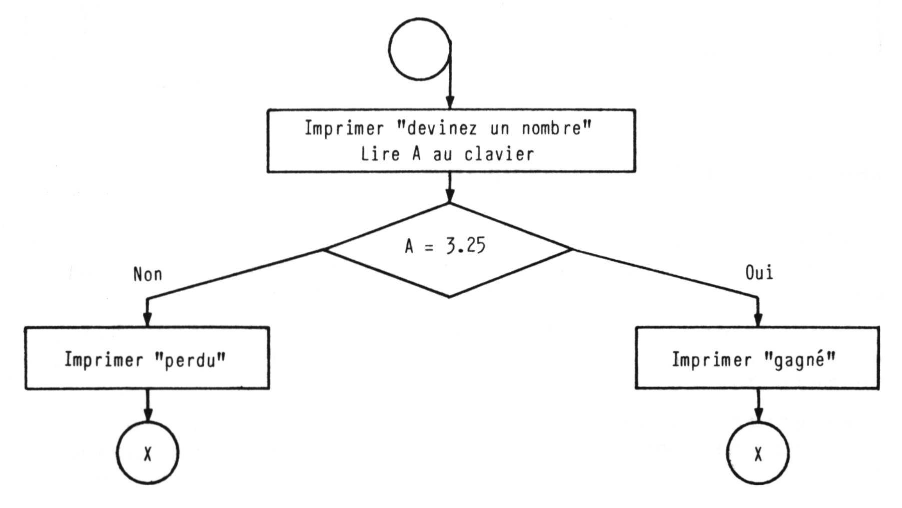
```
20 INPUT "DEVINEZ UN   NOMBRE";A
30 IF A=3.25 THEN PRINT"GAGNE" ELSE PRINT"PERDU"
```

Nous venons maintenant de voir notre première instruction véritablement élaborée. En effet, elle rend l'ORIC capable de prendre des décisions en fonction des différentes situations qui peuvent résulter des données. En fait, c'est vous qui prenez les décisions en préparant le programme, et si jamais vous oubliez un cas possible, le programme se comportera incorrectement si le cas se trouve réalisé.

De décision en décision, le cheminement peut se ramifier de façon complexe.

Les *ordinogrammes*, encore appelés *organigrammes*, peuvent alors être nécessaires pour s'y retrouver.

Avant de passer aux améliorations de notre programme de jeu, nous allons étudier l'ordinogramme du programme B-1.



Il est formé de blocs, qui spécifient les différentes opérations, reliés entre-eux par des flèches qui représentent l'ordre de succession des opérations. La forme même du bloc indique au premier coup d'oeil la nature de l'opération.

Parmi les formes de bloc, on distingue essentiellement :

- *le rectangle*, qui a une seule entrée et une seule sortie et représente toute opération impérative ;
- *le losange*, qui a une seule entrée, mais deux ou plusieurs sorties, et représente les opérations de test.

On utilise les signes spéciaux et (X) pour marquer le début et la fin du traitement.

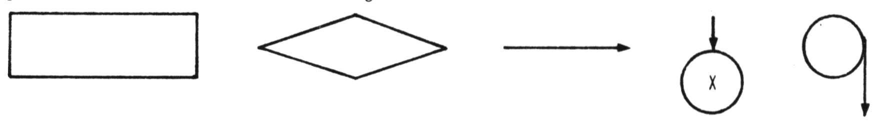
Il est toujours recommandé de tracer l'ordinogramme avant d'entreprendre la rédaction du programme : ce n'est jamais une perte de temps.

Exercice 4.2

Tracez l'ordinogramme du programme A-2. Tracez l'ordinogramme de l'exercice 4.1.

Exercice 4.3

A quels mots-clés Basic, vus jusqu'à présent correspondent les blocs ou signes :



PERFECTIONNEMENTS DU PROGRAMME B

Il faut bien avouer que, dans sa première version, le jeu n'est pas particulièrement intéressant. Mais nous sommes maintenant en mesure de l'améliorer.

La première objection ne sera résolue que tout-à-fait à la fin. C'est dommage, car elle concerne la crédibilité même du jeu.

En effet, il suffit de taper LIST pour avoir le listing du programme, et par là-même prendre connaissance du nombre à deviner. Nous supposerons un certain temps

que le joueur ne connaît pas la commande LIST, ou alors qu'il "joue le jeu".

Il faut noter que cette objection se présente aussi, même dans des programmes plus élaborés : par exemple, si vous jouez à la bataille navale contre l'ordinateur et si vous connaissez bien le programme, vous pouvez faire imprimer les variables qui contiennent les coordonnées des navires adverses.

Supposant maintenant que vous jouez sans tricher, vous allez objecter que le jeu est très difficile, voire non équitable : vous avez peu de chances de trouver le nombre du premier coup, ce qu'exige le programme. Il est clair qu'il faut laisser plusieurs chances au joueur.

Cela peut se faire très simplement : il suffit de remplacer l'instruction 5Ø du programme B-1 par 5Ø GOTO 1Ø et on a maintenant droit à un nombre illimité de tentatives.

Malgré cela, le jeu reste bien hasardeux pour deux raisons :

- en cas d'échec, rien ne dit au joueur s'il est loin ou près du résultat ;
- le joueur doit tomber pile sur la bonne valeur, c'est-à-dire doit fournir le nombre à la précision près à laquelle l'ORIC travaille, soit 10^{-9} près.

Nous résoudrons le premier problème en calculant et en imprimant à chaque fois le pourcentage d'erreur E égal au nombre proposé moins le nombre à trouver, divisé par le nombre à trouver, le tout multiplié par cent.

Le pourcentage d'erreur sera imprimé en valeur absolue, donc sans indiquer le sens de l'erreur, pour laisser une certaine difficulté au jeu.

En ce qui concerne l'erreur possible dûe au manque de précision des PSI on ne comparera pas le nombre proposé au nombre à trouver, mais on considérera que la réponse est exacte si E, pourcentage d'erreur, est inférieur à 0,5 %.

Le problème de précision se pose à chaque fois que l'on a des calculs à effectuer. Pour le résoudre, il suffit de remplacer un test d'égalité pure du type IF A = B par un test sur la valeur de la différence entre les deux nombres, du type A-B < seuil, le seuil étant l'ordre de grandeur de la précision - ou plutôt de

l'imprécision ! - du P.S.I. C'est d'ailleurs en fait la valeur absolue de la différence qu'il faudrait tester.

Notre programme devient :

PROGRAMME B-2

```
20 INPUT"DEVINEZ UN NOMBRE";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT"ERREUR";E;"%"
50 GOTO 20
60 PRINT"GAGNE!"
```

A la ligne 25, nous utilisons la fonction ABS (valeur absolue) qui fait partie de la bibliothèque mathématique du Basic de l'ORIC dont la liste complète est donnée en annexe.

Si nous faisons tourner le programme tel qu'il est, nous obtenons, par exemple, l'affichage :

```
ERREUR 10.7692308 %
```

Il est bien évident que nous n'avons que faire de tant de décimales. Comment les supprimer ? Nous utiliserons une autre fonction de la bibliothèque, la fonction INT, qui prend la partie entière de l'argument cité entre parenthèses.

Dans l'instruction 40, remplaçons E par INT(E). Cette fois nous obtenons un pourcentage entier. Là, c'est trop peu. Comment faire pour garder - disons - deux décimales ?

Pour cela, nous multiplions E par 100 pour faire passer les deux décimales que nous voulons garder dans la partie entière, nous prenons le INT de ce produit, ce qui fait disparaître les autres décimales, puis nous redivisons par 100 :

```
INT(E*100)/100
```

Telle est l'expression qui vient remplacer E dans l'instruction 40 du programme B-2, et nous avons maintenant un affichage satisfaisant.

Exercice 4.4

On veut imprimer le nombre X avec D décimales. Ecrire l'instruction correspondante en mode direct.

Pour constituer la version 3 de notre programme, nous voulons ajouter encore un petit perfectionnement. Il serait agréable, lorsque le nombre à deviner est trouvé,

d'imprimer le nombre de tentatives qui ont été nécessaires. C'est, en somme, le score du jeu. Pour cela, nous introduisons une nouvelle variable N, à laquelle nous ajoutons 1 à chaque fois qu'une tentative est faite sans succès, et que nous imprimons à la fin ; d'où le programme B-3 :

PROGRAMME B-3

```
20 INPUT"DEVINEZ UN NOMBRE";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
35 N=N+1
40 PRINT"ERREUR";INT(E*100)/100;"%"
50 GOTO 20
60 PRINT"GAGNE EN";N+1;"COUPS"
```

L'instruction 35 peut sembler paradoxale, mais n'oublions pas que le signe = n'a pas en Basic le même sens qu'en mathématiques. En Basic, il signifie : calculez l'expression qui est à droite, dont N + 1, et mettez le résultat dans la variable qui est à gauche. Ici, c'est la même variable, ce qui est parfaitement licite, et cela revient bien à incrémenter N.

***Question** : La toute première fois qu'on passe sur l'instruction 35, on doit calculer l'expression N + 1 ; quelle valeur prend-on pour N ?*

Vous avez raison de poser cette question qui soulève le problème de *l'initialisation des variables*. Mais nous savons que lorsque l'on fait RUN, Basic met toutes les variables - dont N - à Ø. Or, lorsque l'on commence, on a fait Ø tentative, donc la valeur initiale automatique de N convient. Si, dans un autre problème, on avait eu besoin d'une autre valeur initiale que Ø, alors il aurait fallu la fournir explicitement dans les premières instructions du programme ; ce point est fondamental : beaucoup de programmes échouent pour initialisation incorrecte de certaines variables.

On remarque enfin, en 6Ø, que l'on imprime N + 1 et non N : c'est pour comptabiliser la dernière tentative, car lorsque le nombre est bon, on ne passe pas sur l'instruction 35.

Exercice 4.5

Comment faire imprimer quand même N en 6Ø ?

LES BOUCLES FOR...NEXT

Nous pouvons maintenant nous attaquer à un autre défaut du programme dans son état actuel : il permet un nombre illimité de tentatives. C'est trop, bien sûr. Il faut autoriser plusieurs tentatives, mais en nombre limité, par exemple 5 ou 10.

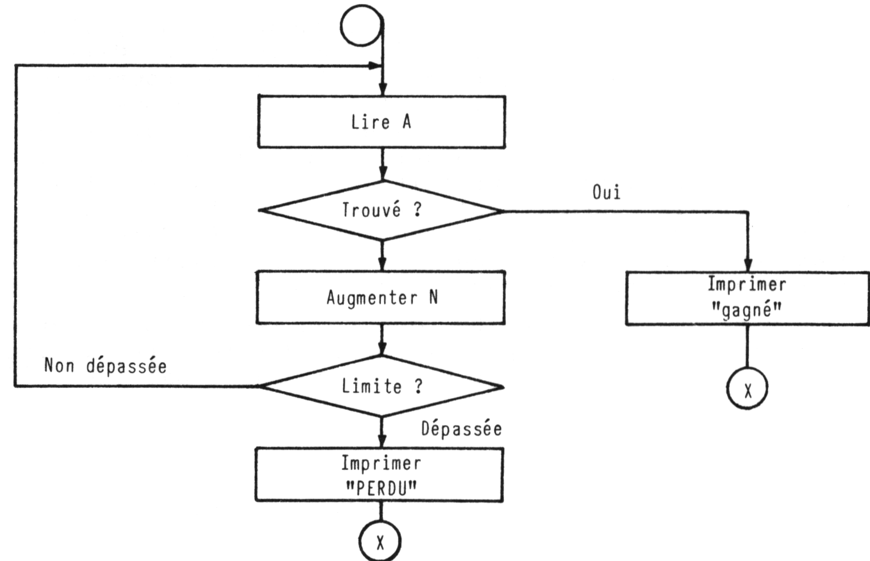
Cela peut se faire très simplement. En effet, nous avons, à tout moment, une mesure du nombre de tentatives faites jusque-là : c'est la variable N ; il suffit de la tester. On remplacera, pour ce faire, l'instruction 5Ø par la séquence :

```
45 IF N<1Ø GOTO 2Ø
5Ø PRINT "JE REGRETTE.VOUS AVEZ PERDU"
55 END
```

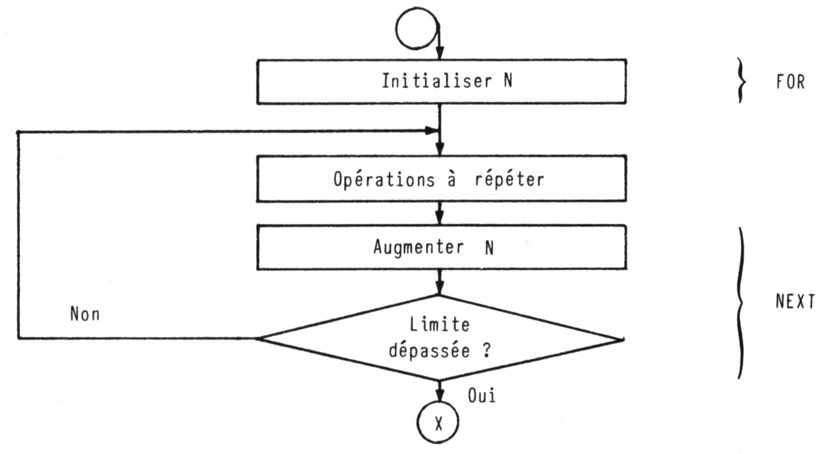
d'où le programme B-4A :

```
20 INPUT"DEVINEZ UN NOMBRE";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT"ERREUR";INT(E*100)/100;"%"
45 IF N<10 GOTO 20
50 PRINT"JE REGRETTE.VOUS AVEZ PERDU"
55 END
60 PRINT"GAGNE EN";N+1;"COUPS"
```

Dessinons l'ordinogramme correspondant sans trop le détailler :



Si nous simplifions encore cet ordinogramme pour en faire ressortir l'ossature, nous obtenons :



Cette structure, très classique et tout à fait fondamentale étant donné son utilisation universelle, s'appelle une boucle. Mais contrairement aux boucles que nous avons vues au début, ici le nombre d'itérations est limité d'avance : N joue le rôle de compteur de passages sur les opérations à répéter ; on lui donne une valeur initiale, puis on effectue le traitement à répéter pour les valeurs successives de N tant que la limite n'est pas dépassée.

Le programme B-4A nous prouve qu'on peut réaliser des boucles, de façon parfaitement satisfaisante, avec les instructions que nous connaissons déjà. Pourtant, étant donné l'importance des boucles, Basic offre un jeu d'instructions spéciales qui permettent de les implanter encore plus facilement. C'est l'ensemble *FOR...NEXT* utilisé dans le programme B-4B.

On voit combien il est facile d'utiliser de telles boucles : il suffit d'encadrer les instructions à répéter (2Ø à 4Ø dans notre exemple) par : *FOR et NEXT*.

PROGRAMME B-4B

```
10 FOR N=1 TO 10
20 INPUT"DEVINEZ UN NOMBRE";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT"ERREUR";INT(E*100)/100;"%"
45 NEXT N
50 PRINT"JE REGRETTE.VOUS AVEZ PERDU"
55 END
60 PRINT"GAGNE EN";N+1;"COUPS"
```

- en tête, une instruction FOR, de la forme

    FOR N = valeur de départ TO valeur limite

ou, en français :

    <u>pour</u> N = valeur de départ <u>jusqu'à</u> valeur limite.

- à la fin, l'instruction NEXT N qui veut dire passer au suivant. Elle incorpore donc à la fois l'incrémentation de N et le test.

Evident, n'est-ce pas ?

Exercice 4.6

Imprimer une table des carrés et des racines carrées des entiers de 1 à 10.

L'instruction à répéter est d'imprimer sur la même ligne, un nombre N, son carré et sa racine, soit :

```
20 PRINT N; N*N; SQR(N)
```

Ceci est à faire pour toutes les valeurs de N de 1 à 10. Soit :

```
10 FOR N = 1 TO 10
```

et on ne doit pas oublier de terminer par :

```
30 NEXT N
```

Exercice 4.7

Implanter les instructions précédentes, et faire exécuter. Qu'est-ce qui manque ?

(Indication : ne concerne pas la boucle).

EXTENSIONS DE FOR...NEXT

La forme que nous venons de voir n'est qu'un cas particulier de la forme plus générale :

    FOR N = valeur de départ TO valeur limite STEP pas

STEP annonce le pas d'incrémentation du compteur. On met 2, par exemple, si l'on veut que N progresse de 2 en 2.

Exercice 4.8

On veut faire la même table qu'à l'exercice 4.6, mais seulement pour les valeurs paires de N.

Si on ne met pas STEP et un pas, l'ORIC sous-entend un pas égal à 1.

Les valeurs des bornes peuvent être quelconques et *le pas peut être négatif*. Par exemple, si on voulait faire la même table que précédemment, mais en commençant par 10, puis 9, 8 etc... on écrirait :

```
10 FOR N = 10 TO 1 STEP -1
```

On n'est pas obligé de mettre des constantes comme bornes : on peut mettre des *variables*, ou même n'importe quelle *expression arithmétique*. Mais les expressions sont évaluées une fois pour toutes lorsque l'on entre dans la boucle, même si l'exécution de la boucle fait évoluer les variables qui interviennent.

Ceci est utilisé surtout dans des écritures de la forme :

```
      FOR I = 1 TO N+1 ...
ou    FOR I = 1 TO N STEP 2*K ...
```

Rien n'oblige les paramètres à être entiers. Par exemple, pour étudier une fonction, on peut être amené à écrire :

```
FOR X = -3.5 TO 4.82 STEP 0.01
```

La *valeur limite* qu'on donne est la valeur qui ne sera pas dépassée pour l'exécution de la boucle ; le test se passe exactement comme sur l'ordinogramme du programme B-4A : le compteur est modifié et on teste s'il est toujours compris entre les bornes ; si oui, on réexécute la boucle, sinon, on a terminé, et le compteur a une valeur hors des bornes.

Exercice 4.9

Pour quelles valeurs du compteur sont exécutées les boucles suivantes, et quelle est la valeur finale du compteur ?

```
10 FOR I = 1 TO 8.5  STEP 2
50 FOR M = 10 TO 3.5 STEP -1
```

Le traitement à répéter dans une boucle peut lui-même contenir une boucle. On dit qu'on a des *boucles imbriquées*.

Exercice 4.10

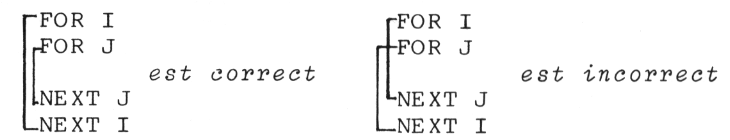
On veut tracer une table des nombres de 1 à 10, avec leurs carrés mais avec 2 couples par ligne.

Une solution est :

```
10 FOR I = 1 TO 9 STEP 2
20 FOR J = 0 TO 1
30 PRINT I+J ; (I+J)*(I+J),
40 NEXT J:PRINT
50 NEXT I
```

le PRINT en 40 assure le passage à la ligne

La deuxième boucle doit toujours être complètement contenue à l'intérieur de la première :



Remarque : Si en 30 on écrivait PRINT I+J ; (I+J)↑2, l'impression serait troublée par le fait qu'on trouverait :

$7^2 = 49.0000001$ et $9^2 = 81.0000001$

Pourquoi ? Eh bien c'est parce que l'interpréteur calcule x^y sous la forme : $e^{y \log x}$ sans faire de cas particulier pour y = 2, d'où des erreurs d'arrondi.

On voit là une des causes d'inefficacité des interpréteurs, alors qu'un programmeur en langage machine tient compte du cas particulier qu'il rencontre, et remplace la puissance 2 par une multiplication, plus rapide.

On peut omettre de répéter la variable qui sert de compteur dans le NEXT. Ainsi, dans l'exercice 4.6, on aurait pu écrire 30 NEXT, et dans 4.10, 40 NEXT... 50 NEXT - et le problème de l'ordre ne se posait plus.

Un couple 40 NEXT J 50 NEXT I peut être remplacé par 45 NEXT J,I (attention à l'ordre).

Avec le couple FOR...NEXT, nous venons d'ajouter à notre arsenal un des outils les plus efficaces de Basic. Il nous reste à voir deux autres éléments de base pour lesquels nous revenons à notre programme de jeu, mais avant, une autre sorte de boucles permises par l'ORIC.

LA BOUCLE REPEAT...UNTIL (REPETER...JUSQU'A)

La structure :
```
10 REPEAT
20 ...
30 ...
40 UNTIL condition
50 ...
```
fait répéter les instructions 20 et 30 (par exemple) jusqu'à ce que la condition écrite en 40 soit satisfaite ; on passe alors à 50.

A la différence de FOR, on ne connaît pas d'avance le nombre d'itérations qu'il faudra pour s'arrêter.

La 'condition' a la même forme que dans IF. Il faut que les instructions entre **REPEAT** et **UNTIL** aient des chances de faire évoluer la condition, sinon on ne s'arrêtera jamais. Si l'on arrive au REPEAT alors que la condition d'arrêt est déjà satisfaite, il y aura quand même une itération d'effectuée.

```
10 FOR I = 1 TO 10 STEP K
50 NEXT
```
peut s'écrire :
```
5 I=1
10 REPEAT
.
.
50 I=I+K
60 UNTIL I>10
```
Il faut faire figurer l'incrémentation explicitement...

Le programme B-1 avec nombre illimité de tentatives peut s'écrire :
```
10 REPEAT
20 INPUT A
30 UNTIL A=3.25
40 PRINT "GAGNE"
```

Bien sûr, on peut se passer de REPEAT...UNTIL puisqu'elle peut être simulée par d'autres instructions comme on l'a vu, mais elle offre une écriture extrêmement parlante. Son emploi a bon escient est donc recommandé.

L'HORLOGE TEMPS REEL

Ce qui manque dans notre jeu actuellement, c'est le "sport". Certes, le joueur a droit à un nombre de tentatives limité, mais ce serait beaucoup plus spectaculaire si le temps alloué pour deviner le bon nombre était limité.

L'ORIC a ce qu'il faut pour cela. En effet, il possède une horloge temps réel, ce qui est intéressant pour sa catégorie de prix.

Mais qu'est-ce qu'une horloge temps-réel, et à quoi ça sert ? Bien sûr, une horloge est faite pour donner l'heure, mais comment procède-t-on ?

Dans le cas de l'ORIC l'horloge se comporte comme une case mémoire qu'on peut lire (c'est de cette façon que l'on obtient l'heure) ; mais cette case mémoire a un comportement un peu particulier : tous les soixantièmes de seconde, son contenu est diminué de 1 par un processus extérieur au microprocesseur, qui fait intervenir un oscillateur, des divisions de fréquence et des interruptions, et dont nous n'avons pas à nous soucier.

Tout ce que nous avons besoin de savoir, c'est qu'il existe un couple de cases-mémoire d'adresses 630 et 631 qui forment un nombre de 16 bits n. A la mise sous tension, ce nombre vaut 65535 et il est décrémenté tous les soixantièmes de seconde (environ).

n=(63Ø)+256*(631) (les parenthèses signifient "contenu de").
A tout instant, 65535-n représente le nombre de soixantièmes de secondes écoulés depuis le dernier passage par zéro. Si l'on désire le temps en secondes, on prend (65535-n)/6Ø.

Oui, mais comment avoir le contenu des adresses 63Ø et 631 ? Basic a ce qu'il faut pour cela. Il offre des moyens de lire ou écrire à toute adresse mémoire que l'on spécifie.

- **PEEK (x)** fournit la valeur comprise entre 0 et 255 de l'octet d'adresse x. x peut être fourni en décimal (Ex. 1ØØØ) ou en hexadécimal (signalé par #, Ex. #3E8)

<u>Exemple</u> : PEEK(Ø) fournit Ø car il y a la valeur Ø à l'adresse Ø de la mémoire.

Si le programme actuellement en mémoire commence par la ligne 1Ø, alors ?PEEK(1283) ou ?PEEK(#5Ø3) donne 1Ø.

On peut ainsi prendre connaissance du contenu de tout emplacement mémoire. Ainsi, pour notre horloge nous aurons :

n = PEEK(63Ø) + 256*PEEK(631)

Mais l'ORIC a encore mieux.

- **DEEK (x)** fournit la valeur combinée :

PEEK(x) + 256*PEEK(x+1)

On en a, en effet souvent besoin : à chaque fois qu'un nombre de 16 bits est rangé en mémoire dans deux octets consécutifs, avec l'octet de poids faible d'abord, comme c'est l'habitude du microprocesseur de l'ORIC, le 6502.

Donc notre temps est donné par n=DEEK(63Ø). Mais il nous faut aussi pouvoir écrire dans la mémoire, pour, entre autres initialiser un intervalle de temps.

- **POKE x,y** écrit à l'adresse x, la valeur y (y doit être compris entre 0 et 255, x entre 0 et 65535).

Exemple : essayez POKE 48ØØØ,65. Cela fait apparaître un A tout en haut de l'écran, car 65 est le code du A et 48000 est la première adresse de la mémoire d'écran.

Si l'on veut mettre une valeur de 16 bits en deux octets consécutifs de la mémoire, on dispose, sur l'ORIC, d'un POKE double :

- **DOKE x,y** met en x (partie basse) et x+1 (partie haute) la valeur sur 16 bits y. x et y doivent être compris entre 0 et 65535 (en fait 65534 pour x afin que x+1 reste inférieur à 65535).

Exemple : essayez DOKE Ø,257 et vérifiez par ?PEEK(Ø); PEEK(1) que vous avez bien 1 et 1 (257=1+256*1).

N.B. Dans les quatre opérations précédentes l'adresse peut toujours être fournie sous forme hexadécimale (#...) Dans DOKE, la donnée y peut l'être, mais pas dans PEEK : il n'y a aucun diagnostic mais l'écriture n'est pas faite.

On a donc un moyen facile d'initialiser l'horloge temps réel pour mesurer un intervalle de temps : DOKE 63Ø,65535.

Si l'on écrit :

```
10 DOKE 630,65535
...
50 T=65535-DEEK(630)
```

alors T est proportionnel au délai écoulé entre l'exécution de 10 et l'exécution de 50 : on a donc une mesure de ce délai (pour l'avoir en secondes, il suffit de diviser T par 60).

On peut, au contraire, générer un délai : supposons qu'entre les instructions 10 et 20 on veuille respecter un délai d'une minute ; on écrit :

```
10 ...
15 DOKE 630,65535
16 IF 65535-DEEK(630)<3600 GOTO 16
20 ...
```

En 15, on fixe l'instant de départ. Ensuite, en 16, 65535-DEEK(630) est le délai écoulé entre 15 et la présente exécution du 16 : il change à chaque fois car le temps passe. Tant que le délai est inférieur à 3600 tierces = 60 secondes = 1 minute, on réexécute 16. Mais le délai avance sans cesse : il finira par être à 1 minute, et alors on passera en 20.

Notez que l'ORIC a un moyen plus simple pour cela : WAIT x produit une attente de x centièmes de seconde. On pouvait donc écrire :

```
16 WAIT 6000
```

C'est donc l'horloge temps réel qui va nous servir pour limiter le temps alloué au joueur.

Comment ? C'est tout simple. On va, au début du jeu, initialiser le temps écoulé à 0 :

```
5 DOKE 630,65535
```

puis, à chaque tentative du joueur, on va tester si le temps n'est pas dépassé :

```
15 IF 65535-DEEK(630)>7200 GOTO 50
```

Exercice 4.11

Quel est le temps alloué au joueur par l'instruction 15 ?

On arrive donc au programme B-5

PROGRAMME B-5

```
5 DOKE 630,65535
10 FOR N=1 TO 10
15 IF 65535-DEEK(630)>7200 GOTO 50
20 INPUT"DEVINEZ UN NOMBRE";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT"ERREUR ";INT(E*100)/100;"%"
45 NEXT N
50 PRINT"JE REGRETTE.VOUS AVEZ PERDU"
55 END
60 PRINT"GAGNE EN ";N;"COUPS ET ";INT((65535-DEEK(630))
                                      /60);"SECONDES"
```

On a également incorporé à la ligne 6Ø l'impression du temps mis par le joueur pour trouver la solution.

Exercice 4.12

Reconstituer l'ordinogramme du programme B-5.

Exercice 4.13

Modifier l'instruction 6Ø pour imprimer le temps au 1/100ème de seconde (comme pour les compétitions de ski).

Il reste un petit perfectionnement à apporter : lorsqu'on imprime "perdu", il serait bon de distinguer si c'est par dépassement de temps ou par trop grand nombre d'essais. C'est le but de l'exercice suivant.

Exercice 4.14

Lorsque le joueur a perdu, imprimer la cause de l'échec, temps ou nombre d'essais.

INSTRUCTION STOP, TOUCHES 'Ctrl' C, COMMANDE CONT

Notre programme de jeu est maintenant arrivé à un bon niveau de complexité. Par conséquent, des difficultés peuvent se présenter pour en effectuer la mise au point. Comment l'ORIC nous aide-t-il à les résoudre ?

Le premier outil est constitué par les messages d'erreur imprimés par l'ORIC en cas de situation anormale. Par exemple, si jamais vous appelez la fonction SQR avec un argument négatif, vous aurez le message :

?ILLEGAL QUANTITY ERROR IN numéro d'instruction.

puis, Ready. est affiché, ce qui indique que l'ORIC est prêt à accepter une commande en mode direct.

La commande directe la plus judicieuse à entrer dans un cas semblable est PRINT certaines variables. En effet, lors d'un arrêt de cette sorte, toutes les variables du programme sont conservées, vous pouvez donc demander leur impression en mode direct. Par exemple, si notre racine carrée à argument négatif dépend d'une variable X, nous taperons ?X. Là, nous voyons que X n'a pas la valeur que nous escomptions. Nous pouvons alors demander le LIST de l'instruction qui calcule X. Nous voyons alors qu'il manque une opération, et nous sommes prêts à corriger l'instruction.

Attention, à partir du moment où nous faisons la moindre modification au programme, les variables ne sont plus conservées. Nous avons donc intérêt à examiner le plus possible de variables avant de commencer les corrections.

Tel est le scénario habituel de correction des erreurs. Les messages d'erreur sont listés en annexe, avec une tentative d'analyse de leurs causes les plus fréquentes.

Le procédé est différent s'il ne se produit aucune erreur suscitant un message alors que les résultats sont faux. Comment faire dans ce cas ? A ce moment, on va subdiviser le programme en petites étapes, entre lesquelles on va insérer des instructions STOP. Par exemple, si un programme a deux étapes, la première de 1Ø à 5Ø et la seconde de 6Ø à 15Ø, on intercalera un 55 STOP.

Lorsqu'on arrivera en 55, l'ORIC imprimera :

```
BREAK IN 55
Ready
```

et s'arrêtera.

Notons déjà que le fait d'obtenir une telle impression signifie que la première étape s'est déroulée jusqu'au bout. Correctement ? Pour le savoir, puisque l'ORIC est arrêté, il vous suffit de demander l'impression directe des variables stratégiques.

Supposons que tout soit correct. Nous voudrions maintenant exécuter la deuxième étape. Eh bien, pour cela,

nous disposons de la commande <u>*CONT*</u>, qui veut dire "continuez, maintenant que j'ai fait ce que je voulais lors de l'arrêt". Attention, vous ne pouvez employer CONT si, au cours de l'arrêt, vous avez modifié le programme.

Il y a encore un cas possible. Supposons que, dans l'exemple ci-dessus, on n'obtienne jamais l'affichage de BREAK IN 55. Cela veut dire que, lors de la première étape, le programme entre dans une boucle sans fin. Comment savoir où on en est ? Il suffit d'appuyer sur les touches 'Ctrl' et C. Cette combinaison simule une instruction STOP dans la ligne en cours d'exécution au moment où l'on appuie : on obtient le message BREAK IN 25 (par exemple). On relance l'exécution par CONT. L'examen de quelques variables et un listing de la zone du programme indiquée par le BREAK IN permettent, en principe, de dépister l'erreur.

Un autre outil de dépannage est tout simplement d'ajouter, à intervalles réguliers, l'impression des principales variables ; il suffira ensuite, lorsque le programme sera au point, de supprimer les instructions d'impression superflues.

LA "TRACE"

Un dernier outil, bien pratique, de dépannage est ce qu'on appelle la "**trace**". Lorsqu'elle est activée dans une portion de programme, à chaque fois que l'on passe sur une ligne Basic, le numéro de cette ligne est systématiquement imprimé. On peut donc facilement suivre par où on passe, donc voir si un test s'effectue bien, ou si l'on boucle. On active la trace par **TRON** et on la desactive par **TROFF**.

<u>Exemple</u> : le programme (idiot) suivant

```
10 TRON
20 REPEAT
30 UNTIL I>10
```

affichera [20] [30] [30] [30]... indéfiniment.
Si vous ajoutez 25 I=I+3 vous aurez :

[20] puis 4 couples [25] [30].

Comme les impressions de trace troublent les impressions, on délimite les portions du programme soumises à la trace en les entourant de TRON...TROFF, qu'on supprimera une fois le programme au point.

RECAPITULATION

Ce chapitre nous a permis de voir les outils de base du programmeur :

- les instructions fondamentales **IF** et **FOR**
- la manipulation de **l'horloge temps réel** de l'ORIC
- quelques **aides à la mise au point** des programmes.

Nous sommes maintenant parés pour aborder les techniques élaborées de programmation.

Question : *Quelle est la différence entre STOP et END ?*

La seule différence est que STOP fait imprimer le message BREAK IN... alors que END ne le fait pas.

On peut également reprendre par CONT après une instruction END.

CHAPITRE V

PROGRAMMATION EVOLUEE

INSTRUCTIONS DATA, READ et RESTORE

Notre programme de jeu va maintenant nous conduire à des techniques plus sophistiquées.

Supposons que l'on veuille jouer à plusieurs. Il nous faut donc maintenant une série de nombres à deviner. En effet, si le joueur N° 3 a vu que les deux précédents avaient à deviner 3.25, il n'aura pas trop à se creuser les méninges !

Le programme B-6A donne une solution. Pour simplifier, nous avons supprimé la limite sur le nombre d'essais autorisés, mais, bien sûr, nous avons laissé la limitation du temps.

Attention : Si vous avez laissé en mémoire le précédent programme B-5, vous avez intérêt à effacer la mémoire avec la commande NEW avant de taper le programme ci-dessous. Nous avons, en effet, renuméroté les lignes et la frappe superposée de B-6A sur B-5 donne une "salade" inintelligible. Cette renumérotation a pour but de redonner un peu d'air au programme pour permettre les futures adjonctions et modifications.

```
        PROGRAMME B-6A

10 READ C
20 DOKE 630,65535
30 IF 65535-DEEK(630)>7200 GOTO 90
40 INPUT"QUEL NOMBRE PROPOSEZ-VOUS";A
50 E=100*ABS(A-C)/C
```

```
60 IF E<0.5 GOTO 110
70 PRINT"ERREUR ";INT(E*100)/100;"%"
80 GOTO 30
90 PRINT"TROP TARD! LAISSEZ LA PLACE AU JOUEUR SUIVANT"
100 GOTO 10
110 PRINT"GAGNE EN ";INT((65535-DEEK(630))/.6)/100;
                                          "SECONDES"
120 GOTO 10
200 DATA 3.25,7.63,2,121,449,0.075,18
210 DATA 5.8,210,78.31,901.5,31,4.18,2.7
```

Deux instructions nouvelles apparaissent dans cette version : *READ* et *DATA*.

DATA sert simplement à spécifier une liste de constantes séparées par des virgules ; l'instruction DATA n'est considérée qu'en relation avec une instruction READ ; sinon elle est "transparente" pour le programme.

On peut placer une instruction DATA n'importe où dans le programme ; lorsque Basic arrive dessus, il n'en fait rien et il passe à l'instruction suivante. Ainsi, 2ØØ aurait pu être numéroté 55, et 21Ø aurait pu être numéroté 75, par exemple. L'exécution serait quand même passée directement de 5Ø à 6Ø et de 7Ø à 8Ø. Les données prises n'auraient pas été altérées. Ce qui compte, c'est l'ordre des différentes instructions DATA et l'ordre des données à l'intérieur d'une même instruction DATA.

Au départ du programme (juste après le RUN),la première donnée du premier DATA sera prise lors du premier READ exécuté. Puis, au fur et à mesure de l'exécution des READ successifs, la deuxième donnée du premier DATA, puis la troisième etc. puis la première donnée du deuxième DATA et ainsi de suite.

Dans notre exemple, comme il y a un READ à chaque nouveau joueur, les nombres successifs à chercher seraient 3.25, puis 7.63, puis 2 etc. soit 14 possibilités différentes.

Que se passe-t-il s'il y a plus de 14 joueurs ? Eh bien il y a une erreur : si l'on essaie un READ alors que la dernière donnée du dernier DATA a été "lue", le message ?OUT OF DATA ERROR IN... est affiché par l'ORIC.

L'instruction *RESTORE* nous fait contourner l'obstacle : elle permet, en effet, de revenir au début des DATA. C'est-à-dire qu'après un RESTORE, un READ obtient de nouveau la première donnée du premier DATA puis... etc. Ici, nous allons donc reparcourir la même série de nombres à deviner tous les 14 joueurs.

Pour cela, nous avons besoin d'une variable J, qui va contenir le numéro de joueur. Quand ce numéro deviendra 14 ou divisible par 14, il faudra faire un RESTORE.

Mais, comment voit-on que X est divisible par Y ? Très simple : si X est divisible par Y, le quotient X/Y est égal à INT(X/Y) puisqu'il est entier. D'où le programme B-6B :

```
        PROGRAMME B-6B

10 J=J+1:PRINT "JOUEUR NO. ";J
20 READ C: DOKE 630,65535
30 IF 65535-DEEK(630)>7200 GOTO 70
40 INPUT"QUEL NOMBRE PROPOSEZ-VOUS";A
50 E=100*ABS(A-C)/C: IF E<0.5 GOTO 90
60 PRINT"ERREUR ";INT(E*100)/100;"%": GOTO 30
70 PRINT"TROP TARD! LAISSEZ LA PLACE AU JOUEUR SUIVANT"
75 IF INT(J/14)=J/14 THEN RESTORE
80 GOTO 10
90 PRINT"GAGNE EN ";INT((65535-DEEK(630))/.6)/100;
                                            "SECONDES"
100 GOTO 75
200 DATA 3.25,7.63,2,121,449,0.075,18, 5.8
210 DATA210,78.31,901.5,31,4.18,2.7
```

Une variante supplémentaire, par rapport au programme B-6A, vient de ce qu'il y a plusieurs instructions à certaines lignes. On peut, en effet, mettre plusieurs instructions par ligne à condition de les séparer par le caractère deux-points (:). Cela rend les programmes plus compacts, mais aussi moins lisibles.

Bien entendu, si la ligne "1" renferme plusieurs instructions, un GOTO 1 conduira à la première, pas au milieu de la ligne ! Donc si une instruction doit être cible d'un GOTO ou d'un IF, elle doit être seule sur sa ligne, ou en tête de ligne.

Autre variante : le nombre 5.8 est passé du début du deuxième DATA à la fin du premier. Cela ne change absolument rien à l'ordre des données qui seront prises en compte.

Exercice 5.1

Une autre méthode pour éviter l'épuisement des données serait d'ajouter, en fin de série, une donnée bidon différente des nombres qu'on veut traiter, par exemple 99999, et que l'on teste : si on la trouve, on fait RESTORE. Réalisez une version du programme qui utilise cette méthode.

DIM ET TABLEAUX

Nous abordons maintenant une importante notion qui va fortement augmenter la puissance de traitement mise à notre disposition.

Nous jouons toujours à plusieurs joueurs, que nous limitons à 14, par exemple. Ce que nous voulons de surcroît c'est qu'une fois que tous les joueurs ont effectué leur partie, un tableau récapitulatif des scores obtenus, en nombre de secondes, soit affiché.

Pour ce faire, il faut introduire une nouvelle variable : SC. On obtient alors le programme suivant :

PROGRAMME B-7A

```
10 FOR J=1 TO 14:PRINT "JOUEUR`NO. ";J
20 READ C: DOKE 630,65535
30 SC=(65535-DEEK(630))/60:IF SC>120 GOTO 70
40 INPUT"QUEL NOMBRE PROPOSEZ-VOUS";A
50 E=100*ABS(A-C)/C: IF E<0.5 GOTO 90
60 PRINT"ERREUR ";INT(E*100)/100;"%": GOTO 30
70 PRINT"TROP TARD! LAISSEZ LA PLACE AU JOUEUR SUIVANT"
75 IF INT(J/14)=J/14 THEN RESTORE
80 GOTO 100
90 PRINT"GAGNE EN ";INT(SC*100)/100;"SECONDES"
100 NEXT J
110 PRINT "SCORE = ";SC:END
200 DATA 3.25,7.63,2,121,449,0.075,18, 5.8
210 DATA210,78.31,901.5,31,4.18,2.7
```

Mais cette solution n'est pas satisfaisante. En effet, ce programme n'imprimera jamais que le score du dernier joueur. Ce dont nous avons besoin, c'est un score pour chaque joueur, c'est-à-dire de 14 variables semblables, attachées chacune à un joueur J.

Basic a un outil pour cela. Il permet, en effet, que SC soit une variable multiple dont SC(1) sera le premier élément, SC(2) le deuxième etc.

Le numéro de l'élément voulu est mis entre parenthèses : il s'appelle l'indice. L'indice peut être une variable : SC(I) est le I ième élément, ou même une expression : SC(3*I-4).

Une telle variable multiple s'appelle un *tableau*. Il faut prévenir Basic du fait qu'une variable est un tableau et du nombre d'éléments (cela pour réserver de la place en mémoire). Cela se fait par une instruction *DIM*.

Pour notre exemple, nous avons : DIM SC(14). En fait, ici on réserve la place pour 15 éléments, car l'indice Ø est utilisable.

Bien entendu, l'instruction DIM doit être exécutée avant toute opération sur la variable concernée ; en revanche, elle ne doit être exécutée qu'une fois.

L'instruction DIM n'est pas nécessaire tant que la valeur maximum de l'indice ne dépasse pas 10 : en effet, l'ORIC réserve automatiquement la place pour 10. Si la dimension du tableau doit être plus petite que 10, l'instruction DIM est quand même utile car elle libère de la place.

Plusieurs tableaux peuvent être dimensionnés à l'aide d'une même instruction : DIM A(25), B(5Ø).

La valeur maximum assignée à l'indice peut être une variable (qui vient d'être initialisée) : DIM A(N).

Utilisant ces propriétés, nous arrivons au programme B-7B

PROGRAMME B-7B

```
5 DIM SC(14)
10 FOR J=1 TO 14:PRINT "JOUEUR NO. ";J
20 READ C: DOKE 630,65535
30 SC(J)=(65535-DEEK(630))/60:IF SC(J)>120 GOTO 70
40 INPUT"QUEL NOMBRE PROPOSEZ-VOUS";A
45 SC(J)=(65535-DEEK(630))/60
50 E=100*ABS(A-C)/C: IF E<0.5 GOTO 90
60 PRINT"ERREUR ";INT(E*100)/100;"%": GOTO 30
70 PRINT"TROP TARD! LAISSEZ LA PLACE AU JOUEUR SUIVANT"
75 IF INT(J/14)=J/14 THEN RESTORE
80 GOTO 100
90 PRINT"GAGNE EN ";INT(SC(J)*100)/100;"SECONDES"
100 NEXT J
110 PRINT "JOUEUR  SCORE" :PRINT
120 FOR I=1 TO 14
130 PRINT "    ";I,INT(SC(I)*100)/100 :NEXT I
200 DATA 3.25,7.63,2,121,449,0.075,18, 5.8
210 DATA210,78.31,901.5,31,4.18,2.7
```

L'instruction 45 permet de comptabiliser le temps d'hésitation à répondre à l'INPUT.

Nous laissons maintenant notre jeu de côté pour quelques instants pour voir plusieurs compléments sur les tableaux. Vous avez tout intérêt à sauvegarder ce programme sur cassette grâce à l'ordre CSAVE. Ensuite vous pourrez effacer la mémoire par NEW, votre ORIC sera disponible pour quelques exercices.

GARNISSAGE D'UN TABLEAU

L'instruction d'entrée INPUT peut être mise dans une boucle du type : FOR I=1 TO 1Ø :INPUT A(I) :NEXT I.

On aimerait bien savoir à chaque instant quel élément doit être introduit et avoir des messages imprimés tels que :

```
A(1)?
A(2)?  etc.
```

Comme le message comprend un élément variable (la valeur de l'indice), la forme INPUT "TEXTE";... ne convient pas. Il faut utiliser un PRINT qui se termine par point-virgule (;) pour taper la valeur sur la même ligne :

```
1Ø FOR I = 1 TO 1Ø
2Ø PRINT "A(";I;")";
3Ø INPUT A(I)
4Ø NEXT I
```

SOMME ET MOYENNE DES ELEMENTS D'UN TABLEAU

Les éléments d'un tableau peuvent représenter les différentes observations statistiques d'une grandeur, par exemple les tailles des différents élèves d'une classe. La première opération statistique à effectuer sur une distribution est de calculer sa moyenne ; pour cela, il faut d'abord en calculer la somme.

Pour calculer cette somme, nous utiliserons une variable S initialisée à Ø et à laquelle, dans une boucle, seront ajoutés successivement chacun des éléments :

```
1Ø DIM A(N)
2Ø REM NORMALEMENT INTERVIENT ICI LA LECTURE
                                   DES ELEMENTS
3Ø S=Ø
4Ø FOR I=1 TO N
5Ø S=S + A(I)
6Ø NEXT I
7Ø M=S/N
8Ø PRINT "SOMME= ";S,"MOYENNE= ";M
```

L'initialisation de S à Ø est effectuée en 3Ø ; cette initialisation est superflue en début de programme ; elle peut être nécessaire si on arrive en 3Ø après avoir fait d'autres opérations.

En 2Ø apparaît une nouvelle instruction : *REM* (abréviation de remarque). Elle n'influe aucunement sur la marche du programme, mais elle permet d'y incorporer des commentaires explicatifs, ce qui est souvent utile (bien entendu, ces commentaires consomment de la place mémoire).

Exercice 5.2

Calculer la variance de l'ensemble des éléments A ci-dessus.

$$V = \frac{\Sigma_i (Ai-M)^2}{N-1}$$

Les tableaux sont particulièrement indiqués pour représenter des vecteurs d'un espace vectoriel sur K. Par exemple A(1), A(2), A(3) seront les trois composantes du vecteur $\vec{A}$ dans l'espace à trois dimensions. On verra plus loin que Basic permet aussi la manipulation de matrices.

Exercice 5.3

Etant donné les deux vecteurs U et V d'un espace à N dimensions, calculer leur produit scalaire ($\Sigma u_i v_i$).

NOMBRES AU HASARD - FONCTION RND

Nous sommes maintenant arrivés au moment de résoudre un des problèmes qui nous avait le plus préoccupé au début de notre jeu : comment faire pour que, même s'il triche, le joueur ne puisse trouver d'avance le nombre à deviner ?

Le mieux est que l'ordinateur ne connaisse pas lui-même ce nombre à l'avance, c'est-à-dire qu'il le tire au sort au moment de l'utiliser.

Oui, mais comment un ordinateur dont le comportement doit être le plus déterministe et le plus prévisible possible peut-il donner des nombres aléatoires ?

A priori, cela semble nuisible. Eh bien, et c'est paradoxal, il existe des algorithmes qui font appel à des calculs bien déterminés, qui donnent ce que l'on appelle des séries pseudo-aléatoires, c'est-à-dire des suites de nombres bien déterminés, mais ayant des propriétés

statistiques telles que l'on puisse considérer que tout se passe comme si les nombres avaient été tirés au hasard.

Dans ce contexte, le terme "nombre au hasard" ne peut s'appliquer à un nombre isolé, c'est seulement au niveau d'une suite (nombreuse) de nombres qu'il a un sens.

A quoi peuvent bien servir de telles suites ? Elles servent pour des calculs de simulation dans lesquels il faut tenir compte de phénomènes aléatoires. Par exemple, supposons que l'on veuille simuler dix ans d'exploitation d'une propriété agricole. Un des éléments intervenant peut être la production de blé d'un certain champ, on sait que, quoi qu'il arrive, cette production est comprise entre 10 t (année aux conditions atmosphériques défavorables, mauvaises graines etc.) et 15 t (année réunissant exceptionnellement tous les facteurs favorables).

Pour simuler les aléas dûs à des causes mal connues, le mieux est, pour chaque année, de tirer au hasard un nombre entre 10 et 15 : ce sera le meilleur moyen d'obtenir, dans notre simulation sur dix ans, un certain nombre de bonnes années et de mauvaises années, représentatif de la réalité.

En fait, ici, on ne tirera pas un nombre réparti uniformément entre 10 et 15 : on cherchera à reproduire une loi de probabilité obtenue par observations ou répondant à un modèle.

On voit que les nombres au hasard peuvent être très utiles. L'ORIC peut nous en fournir. Il suffit de faire Y=RND(X) pour obtenir un nombre au hasard compris entre Ø et 1.

Si X est > Ø, on génère différentes séquences pseudo-aléatoires : des appels successifs avec la même valeur de X>Ø, donnent les éléments successifs d'une même suite (le nombre obtenu change à chaque appel, il fait partie de la même suite). En changeant la valeur de X, on change de suite.

Le plus souvent, on utilisera RND(1).

Le nombre obtenu étant compris entre Ø et 1, c'est-à-dire de la forme Ø.232745, il faudra lui faire subir une transformation pour répondre à notre problème. D'une façon générale, il faut obtenir une série de nombres Y comprise entre deux valeurs A et B. L'expression suivante permet d'obtenir ces nombres :

```
Y= A+(B-A)*RND(1)
```

Appliquons ceci dans le programme B-8 pour obtenir un nombre compris entre Ø et 1ØØ

PROGRAMME B-8

```
5 INPUT"NOMBRE DE JOUEURS ";N :DIM SC(N)
10 FOR J=1 TO N:PRINT "JOUEUR NO. ";J
20 C=1+99*RND(1) :DOKE 630,65535
30 SC(J)=(65535-DEEK(630))/60:IF SC(J)>120 GOTO 70
40 INPUT"QUEL NOMBRE PROPOSEZ-VOUS";A
45 SC(J)=(65535-DEEK(630))/60
50 E=100*ABS(A-C)/C: IF E<1 GOTO 90
60 PRINT"ERREUR ";INT(E*100)/100;"%": GOTO 30
70 PRINT"TROP TARD! LAISSEZ LA PLACE AU JOUEUR SUIVANT"
80 SC(J)=0 :GOTO 100
90 PRINT"GAGNE EN ";INT(SC(J)*100)/100;"SECONDES"
95 SC(J)=1+INT((120-SC(J))/24)
100 NEXT J
110 PRINT "JOUEUR  SCORE" :PRINT
120 FOR I=1 TO N
130 PRINT "    ";I,INT(SC(I)*100)/100 :NEXT I
```

Le changement le plus notable est donc dans l'instruction 2Ø où le nombre à chercher est maintenant tiré au hasard. On pourrait le rendre encore plus aléatoire (en effet, au départ du jeu, on aura toujours la même série de nombres) en l'indexant sur le temps, en écrivant, par exemple :

```
7 T=65535-DEEK(63Ø);X= T-1ØØ*INT(T/1ØØ)
2Ø C=1+99*RND(1+X) : DOKE 63Ø,65535
```

La ligne 7 a pour effet de mettre dans X les deux chiffres de droite du temps. Si T = 2873, INT (T/1ØØ) est égal à 28 et X sera égal à 2873-2800 soit 73.

Dans ces conditions, selon le temps écoulé depuis la mise sous tension, une série différente sera générée. D'autres améliorations ont été introduites et ont légèrement facilité le jeu. Par exemple, on a mis la barre à 1 %, mais vous avez toute liberté de jouer sur cette barre et sur la limite de temps.

Le nombre N de joueurs est désormais variable ;
Le score est donné en nombre de points allant de Ø à 5 (Ø si le joueur n'a pas trouvé dans les temps, 5 s'il a trouvé très vite).

Exercice 5.4

Mesurer la qualité statistique du générateur de nombres au hasard de l'ORIC.

Pour cela, nous allons tirer 1ØØØ nombres au hasard, compris entre -1 et +1. Calculons ensuite moyenne et variance (valeurs idéales Ø et $\frac{1}{3}$) ainsi que les effectifs des 1Ø classes :

$[-1,-\frac{4}{5}];[-\frac{4}{5},-\frac{3}{5}]...[Ø,\frac{1}{5}]...[\frac{4}{5},1]$ (valeur idéale 1ØØ).

Le programme suivant est une des solutions possibles :

```
1Ø FOR I=1 TO 1ØØØ
2Ø N=(-1)+2*RND(1)
3Ø S=S+N:S2=S2+N↑2
4Ø J=1+INT((N+1)*5)
5Ø C(J)=C(J)+1
6Ø NEXT I
7Ø M=S/1ØØØ:V=(S2-1ØØØ*M↑2)/999
8Ø PRINT"MOYENNE =";M,"VARIANCE =";V
9Ø PRINT"CLASSES"
1ØØ FOR I=1 TO 1Ø:PRINT C(I);:NEXT
```

Attention, il ne se passe rien pendant 1 minute et demie. Soyez patient !...

TABLEAUX MULTIDIMENSIONNES

Perfectionnons notre jeu, en permettant à chaque joueur de disputer plusieurs parties et en affichant les scores de chacune de ces parties.

L'ORIC permet de réaliser des tableaux à double-entrée (dits aussi : tableaux à deux indices, ou à deux dimensions).

Pour ce faire, on utilise une instruction DIM de la forme :

```
DIM SC(NJ,NP)
```

s'il y a NJ joueurs et NP parties. Le score du joueur J à la partie P sera désigné par SC(J,P); d'où le programme B-9 page suivante.

PROGRAMME B-9

```
10 INPUT"NOMBRE DE JOUEURS ";NJ
20 INPUT"COMBIEN DE PARTIES ";NP
30 DIM SC(NJ,NP)
40 FOR P=1 TO NP:FOR J=1 TO NJ
50 PRINT "JOUEUR NO. ";J,"PARTIE ";P
60 C=1+99*RND(1) :DOKE 630,65535
70 SC(J,P)=(65535-DEEK(630))/60:IF SC(J,P)>120 GOTO 110
80 INPUT"NOMBRE PROPOSE ";A
85 SC(J,P)=(65535-DEEK(630))/60
90 E=100*ABS(A-C)/C: IF E<1 GOTO 130
100 PRINT"ERREUR ";INT(E*100)/100;"%": GOTO 70
110 PRINT"TROP TARD! LAISSEZ LA PLACE AU JOUEUR SUIVANT"
120 SC(J,P)=0 :GOTO 150
130 PRINT"GAGNE EN ";INT(SC(J,P)*100)/100;"SECONDES"
140 SC(J,P)=1+INT((120-SC(J,P))/24)
150 NEXT J,P :PRINT :PRINT
160 PRINT "JOUEUR ";:FOR P=1 TO NP
170 PRINT"PARTIE ";P;:NEXT P :PRINT :PRINT
180 FOR J=1 TO NJ :PRINT "    ";J,
190 FOR P=1 TO NP :PRINT "    ";SC(J,P),
200 NEXT P:PRINT :NEXT J
```

Ces tableaux rectangulaires s'appellent, en mathématiques, des matrices dont les éléments sont répartis en lignes et colonnes. Les matrices ont de nombreuses applications : celles-ci sont donc réalisables sur l'ORIC.

Exercice 5.5

Ecrire un programme qui calcule le produit C de deux matrices A et B (rappel de la formule de définition : $C_{ij}=\sum_k a_{ik}\cdot b_{kj}$*).*

Ce qui vient d'être exposé concernant les tableaux se généralise : le nombre de dimensions peut être quelconque ; à trois dimensions, les éléments sont répartis en plans, lignes et colonnes. La seule restriction formelle au nombre de dimensions est que l'instruction DIM doit tenir en 80 caractères. Mais d'autres limitations interviennent auparavant : la taille mémoire restant disponible ne doit pas être dépassée.

MANIPULATION DES CHAINES DE CARACTERES

Il nous reste encore une amélioration à apporter à notre programme. En effet, il serait souhaitable que, dans l'impression des résultats, ce soit le nom de chaque joueur qui soit écrit, et non son numéro.

Nous allons voir que cela est possible. Il est indispensable, en effet, que l'ordinateur permette de manipuler des textes ou des noms. Il faut bien, en gestion, manipuler le nom des clients !

L'ORIC admet deux sortes de variables : les variables numériques, que nous connaissons déjà, et les variables alphanumériques, ou chaînes de caractères. Le nom d'une variable chaîne se forme comme celui d'une variable numérique, en ajoutant un $ à la fin :

A : variable numérique ;
A$: variable alphanumérique.

Concurremment, dans le même programme, les variables distinctes A et A$ peuvent être utilisées.

Seuls les deux premiers caractères du nom d'une variable sont pris en compte (à part le $) : AL$ et ALFA$ sont la même variable.

Il est possible de former des tableaux de chaînes de caractères DIM NOM$(N) réserve un tableau de N chaînes de caractères, chacun des éléments pouvant comporter un nombre variable de caractères.

Pour affecter une valeur à une chaîne de caractères, l'instruction la plus simple est l'affectation classique : A$="BONJOUR".

Notez que la valeur attribuée est entourée de guillemets.

On peut aussi "lire" la variable au clavier par INPUT A$. Dans ce cas, il n'est pas nécessaire de mettre BONJOUR entre guillemets, car l'ORIC s'attend à une chaîne de caractères.

On peut enfin utiliser READ et DATA. Dans le DATA, les chaînes sont normalement sans guillemets, sauf si elles contiennent un caractère "spécial" comme espace, virgule ou deux points :

```
10 READ A$,B$
20 DATA BONJOUR,"AU REVOIR"
```

Nous sommes maintenant "parés" pour comprendre le programme B-10.

PROGRAMME B-10

```
10 INPUT"NB.DE JOUEURS,NB.DE PARTIES ";NJ,NP
20 DIM NOM$(NJ),SC(NJ,NP)
30 FOR J=1 TO NJ:PRINT"NOM DU JOUEUR NO. ";J;
35 INPUT NOM$(J) :NEXT J
40 FOR P=1 TO NP:FOR J=1 TO NJ
50 PRINT "JOUEUR NO. ";J;"(";NOM$(J);")","PARTIE ";P
60 C=1+99*RND(1) :DOKE 630,65535
70 SC(J,P)=(65535-DEEK(630))/60:IF SC(J,P)>120 GOTO 110
80 INPUT"NOMBRE PROPOSE ";A
65 6C(J,P)=(65535-DEEK(630))/60
90 E=100*ABS(A-C)/C: IF E<1 GOTO 130
100 PRINT"ERREUR ";INT(E*100)/100;"%": GOTO 70
110 PRINT"TROP TARD! LAISSEZ LA PLACE AU JOUEUR SUIVANT"
120 SC(J,P)=0 :GOTO 150
130 PRINT"GAGNE EN ";INT(SC(J,P)*100)/100;"SECONDES"
140 SC(J,P)=1+INT((120-SC(J,P))/24)
150 NEXT J,P :PRINT :PRINT
160 PRINT "JOUEUR ";:FOR P=1 TO NP
170 PRINT"PARTIE ";P;:NEXT P :PRINT :PRINT
180 FOR J=1 TO NJ :PRINT NOM$(J),
190 FOR P=1 TO NP :PRINT "   ";SC(J,P),
200 NEXT P:PRINT :NEXT J
```

Dans le programme précédent, nous ne faisons que lire un nom, le ranger dans NOM$(J) pour le mémoriser et l'imprimer un peu plus tard. C'est souvent le seul traitement à effectuer sur les chaînes de caractères.

Mais, dans certains cas, on doit effectuer des traitements sur les chaînes, comme des comparaisons, des extractions, des conversions etc. Nous voyons maintenant les opérations de ce genre disponibles sur l'ORIC.

Comparaison : Une instruction telle que IF A$=B$ GOTO ... permet de comparer les deux chaînes A$ et B$. Les applications sont nombreuses : on peut, par exemple, vérifier si un mot appartient à un dictionnaire ou si un nom figure dans une liste de personnes autorisées... Une autre comparaison, telle que IF A$<B$... est également utile : en effet, le mot A$ est considéré comme inférieur à B$ s'il le précède dans l'ordre alphabétique, d'où un moyen de classer des listes de noms par ordre alphabétique.

Concaténation : L'opérateur + appliqué à deux chaînes de caractères produit leur juxtaposition :

```
"BON" + "JOUR"      fournit BONJOUR. Tapez :
A$ = "NJ"
B$ = "UR"
```

```
C$ = "BO"+A$+"O"+B$
?C$                    là encore,vous obtenez BONJOUR
```

Chaîne vide : C'est la chaîne formée de 0 caractère. A$="" fournit à A$ la valeur 'chaîne vide' ; quel que soit X$, X$+A$ sera identique à X$.

Extraction de sous-chaînes : L'ORIC possède un certain nombre de fonctions permettant la "manipulation" des chaînes de caractères. Si le nom de la fonction se termine par $, son résultat est une chaîne de caractères ; dans le cas contraire, ce résultat est un nombre.

- *LEN(X$)* : fournit la longueur (nombre de caractères) de la chaîne ;
- *LEFT$(X$,N)* : fournit les N caractères les plus à gauche, extraits de la chaîne X$;
- *RIGHT$(X$,N)* : fournit les N caractères les plus à droite, extraits de la chaîne X$. Si N>LEN(X$), on obtient toute la chaîne (valable aussi pour LEFT$) ;
- *MID$* : extrait des caractères au milieu d'une chaîne ; elle peut avoir deux ou trois arguments ;
- *MID$((X$,K)* : fournit les caractères extraits de la chaîne X$ à partir de la position K. Si K>LEN(X$), on obtient la chaîne vide ;
- *MID$(X$,K,N)* : fournit la sous-chaîne de N caractères extraits de X$ à partir du K ième. Si K>LEN(X$), on obtient la chaîne vide ; si N spécifie plus de caractères qu'il n'en reste dans X$, on obtient tous les caractères de X$ à partir du K ième.

Exercice 5.6

Remplacer le 5ème caractère de la chaîne X$ par la lettre A.

Exercice 5.7

Vérifier si la chaîne A$ contient la sous-chaîne B$. Renvoyer dans la variable K, Ø si A$ ne la contient pas, et si elle la contient, la première position dans A$ où on trouve B$

Par exemple : si B$="BRA", dans ABRACADABRA, on trouve B$ en 2 et en 9, et on doit obtenir K=2. Avec BONJOUR, on doit obtenir K=Ø.

Fonctions de conversion : Les autres fonctions chaînes de caractères effectuent des conversions entre les nombres et leur représentation sous forme de chaînes de caractères ou de code ASCII (le code ASCII est le mode de représentation interne choisi par l'ORIC et la plupart des micro-ordinateurs : chaque caractère est représenté par un motif binaire choisi, occupant un octet).

Il y a quatre fonctions de ce type :

- *ASC(X$) :* fournit la valeur en décimal de l'octet qui représente en ASCII le premier caractère de X$. Exemple : ?ASC("A") donne 65.
- *CHR$(K) :* fournit une chaîne de un caractère : le caractère dont K est le code ASCII. Exemple : ?CHR$(65) fait imprimer un A.
- *STR$(A) :* fournit la chaîne de caractères qui est la représentation en décimal du nombre A. Si A=3.5, STR$(A) est la chaîne : signe (1), chiffre 3, point, chiffre 5. ?A et ?STR$(A) produisent la même impression à ceci près que ?A est suivi d'un mouvement de curseur à droite et non ?STR$(A). Observez la différence entre :

  ?"AA";A;"AAA"   et   ?"AA";STR$(A);"AAA"

Il y a néanmoins une différence importante : STR$(A) est une chaîne de caractères justiciable des opérations LEFT$...

- *VAL(X$) :* fournit la valeur du nombre dont X$ est la représentation décimale. Les seuls caractères permis dans X$ sont les chiffres, le point, l'espace et + ou -. Si le premier caractère non blanc de X$ n'est pas l'un des caractères permis, on obtient 0.

Exercice 5.8

A est entier positif. Trouvez son nombre de chiffres. Même question si A est entier quelconque.

Exercice 5.9

Imprimez B en supprimant 3 décimales.

---

(1) Pour un nombre négatif, on a le signe -. Pour un nombre positif(c'est un défaut du Basic de l'ORIC), on a le caractère de code ASCII 2. Il ne s'imprime pas.

RECAPITULATION

Nous sommes maintenant arrivés à un état perfectionné de notre programme de jeu.

Au passage, nous avons découvert les principales possibilités de Basic.

Les instructions décrites jusqu'ici sont en général valables pour tous les ordinateurs.

Nous allons voir maintenant des propriétés particulières de l'ORIC qui, pour la plupart, ne se retrouvent pas sur les autres P.S.I., notamment les possibilités graphiques et sonores.

Mais, auparavant, nous voyons deux exercices essentiels.

Exercice 5.10

Calculez le score moyen de chaque joueur SM(J). Imprimez le nom et le score moyen du joueur qui a le score maximum. En cas d'ex-aequo, on prend le premier trouvé.

Il est particulièrement nécessaire ici de raisonner sur l'ordinogramme pour étudier le problème du maximum. Ce problème très classique se pose dans d'innombrables applications.

Exercice 5.11

Imprimez le classement des joueurs, et non pas seulement le premier.

Rappel : Des solutions aux exercices sont proposées en annexe.

CHAPITRE VI

COURBES, GRAPHIQUES HAUTE RESOLUTION

L'instruction la plus simple permettant de faire des dessins sur l'écran, nous la connaissons déjà. C'est tout simplement l'instruction **PRINT**.

PRINT "chaîne de caractères" fait imprimer une chaîne de caractères sur l'écran. Cela permet d'imprimer un texte, comme nous l'avons déjà vu. Nous allons voir maintenant comment cela nous permet aussi de tracer sur l'écran la courbe représentative d'une fonction, programme qui a des applications importantes.

COURBE REPRESENTATIVE D'UNE FONCTION - INSTRUCTION DEF FN

Le programme n'est, en fait, pas très compliqué. Nous allons tracer la courbe de la fonction exponentielle y = exp(x), (e^x) pour x allant de Ø à 1. La courbe sera tracée point par point. Temporairement, l'axe des x sera vertical, l'axe des y sera horizontal. Disposant de 25 lignes sur l'écran, on pourra représenter 24 points.

Sur la ligne i sera matérialisé le point correspondant à $x = \frac{i}{24}$. L'axe des abcisses est ici vertical, orienté de gauche à droite. y peut, lui, varier de 1 à 38. Sur une ligne donnée, une étoile sera imprimée dans la colonne dont le numéro est proportionnel à la valeur de y pour le x correspondant à la ligne.

Il faut donc appliquer un facteur d'échelle à la fonction, pour qu'elle varie entre 1 et 38 afin de couvrir tout l'écran.

Pour la fonction exp, on emploiera :

$$y = \frac{37 \cdot (\exp(x)-1}{e - 1} + 1$$

La formule générale pour une fonction f(x) dont le maximum et le minimum dans l'intervalle de variation étudié sont respectivement max et min serait :

$$y = 1 + \frac{37\ (f(x)-min)}{max - min}$$

Pour imprimer la courbe, il suffit donc de réaliser une boucle FOR X = début TO fin STEP intervalle, et pour chaque point une étoile dans la colonne INT(y).

Basic dispose de deux fonctions simples pour cela :

PRINT TAB(N); positionne à la colonne N
PRINT SPC(N); fait imprimer N espaces donc positionne en N+1

Le programme à réaliser en découle directement :

PROGRAMME C-1A

```
20 FOR X=0 TO 1 STEP 1/24
30 N=1+INT(37*(EXP(X)-1)/(EXP(1)-1))
40 PRINT SPC(N);"*"
50 NEXT
```

Bien entendu, un programme véritablement opérationnel devrait offrir une meilleure présentation : titre, dessin des axes, coordonnées, etc.

Vous avez toutes les connaissances utiles pour le faire, et nous vous suggérons de vous y exercer.

Exercice 6.1

Tracez deux courbes sur le même graphique, par exemple sin(x) et cos(x) pour x de 0 à 2PI.

<u>*Indication*</u> *: ?CHR$(11) fait remonter l'impression à la ligne précédente.*

Nous traitons l'exercice dans le texte vu les difficultés qu'il pose et les solutions apportées.

Compte-tenu de l'indication donnée, une solution simple est :

EX 6-1A

```
10 CLS
20 FOR X=0 TO 2*PI STEP 2*PI/22
30 N1=1+INT(36*(SIN(X)+1)/2)
35 N2=1+INT(36*(COS(X)+1)/2)
40 PRINT SPC(N1);"*"
45 PRINT CHR$(11);SPC(N2);"+"
50 NEXT X
```

La 2ème courbe s'inscrit avec des +. On a réduit l'intervalle d'écriture à 23 lignes (d'où le 22 à l'instruction 2Ø) et l'intervalle de variation à 37 (d'où le 36 en 3Ø et 35) pour que les courbes tiennent bien dans l'écran.

Mais il y a un défaut : les SPC(N2) de la 2ème courbe effacent la première courbe là où y1<y2. Il faudrait tester pour imprimer le plus grand d'abord, ce qui complique le programme.

En fait, au lieu d'espaces il faudrait des curseurs à droite. De même que PRINT CHR$(11) équivaut à un curseur en haut, on a les codes suivants :

CHR$(...)	*Mouvement*
8	curseur à gauche
9	curseur à droite
10	curseur bas
11	curseur haut
12	vidage écran
13	retour chariot
30	retour en haut et à gauche de l'écran (sans effacement)
32	espace

Nous allons donc nous constituer une chaîne de curseurs à droite : D$="":FOR I=1 TO 4Ø:D$=D$+CHR$(9):NEXT et pour faire N curseurs à droite, il suffira de prendre LEFT$(D$,N).

D'où la seconde version :

EX 6-1B

```
10 CLS:H$=CHR$(11):D$=""
15 FOR I=1 TO 40:D$=D$+CHR$(9):NEXT
20 FOR X=0 TO 2*PI STEP 2*PI/22
```

```
30 N1=1+INT(36*(SIN(X)+1)/2)
35 N2=1+INT(36*(COS(X)+1)/2)
40 PRINT LEFT$(D$,N1);"*"
45 PRINT H$;LEFT$(D$,N2);"+"
50 NEXT X
```

INSTRUCTION DEF FN

Il peut arriver de devoir tracer la courbe d'une fonction qui n'est pas exactement une des fonctions de la bibliothèque, mais une combinaison de plusieurs d'entre elles.

Cette combinaison peut intervenir plusieurs fois dans le programme. Il est alors pénible de la réécrire à chaque fois.

Il existe une instruction qui permet de résoudre ce problème : c'est une instruction qui permet à l'utilisateur de définir ses propres fonctions, qui vont alors se rajouter aux fonctions de bibliothèque.

Exemples :

```
10 DEF FNHARM(X)=(A*X+B)/(C*X+D)
20 DEF FNA(X)=2*SIN(X)+COS(2*X)
30 DEF FNF(T)=EXP(-T↑2/2)
40 DEF FNG(X)=1 + 36*(FNF(X)-MIN)/(MAX-MIN)
```

Le nom de la fonction définie suit FN ; il obéit aux règles habituelles applicables aux noms de variables.

Un appel à la fonction FNHARM définie en 10 pourrait être 100 Z= 1+FNHARM(3.5)

L'expression donnée dans la définition est alors calculée, en remplaçant l'argument formel par la valeur (ici 3.5) fournie lors de l'appel. Pour les autres variables qui interviennent (ici A, B, C et D), ce sont leurs valeurs au moment de l'appel qui sont prises en compte.
L'argument peut être fourni lui-même sous forme d'une expression quelconque à calculer :

```
100 PRINT FNA(U*A + EXP(B)/C)
```

Les exemples 30 et 40 ci-dessus montrent qu'il est possible d'utiliser, dans la définition d'une fonction, une fonction déjà définie.

Utilisant cette nouvelle possibilité, voici le programme C-1B qui trace la courbe de la fonction de Gauss entre A et B :

PROGRAMME C-1B

```
5 CLS:INPUT"BORNES";A,B
10 DEF FNF(X)=EXP(-X^2/2)
20 DEF FNG(X)=1+36*(FNF(X)-MIN)/(MAX-MIN)
30 MAX=-10^38:MIN=10^38
40 FOR X=A TO B STEP (B-A)/22
50 IF FNF(X)>MAX THEN MAX=FNF(X)
60 IF FNF(X)<MI THEN MIN=FNF(X)
70 NEXT X
80 FOR X=A TO B STEP (B-A)/22
90 N=INT(FNG(X))
100 PRINT SPC(N);"*":NEXT X
110 GOTO 110
```

La dernière ligne évite l'impression de Ready.

L'avantage du programme C-1B sur la version précédente est qu'il est général : les bornes sont fournies par INPUT, et pour changer de fonction à tracer, il suffit de retaper l'instruction 1Ø.

De 3Ø à 7Ø s'effectuent les calculs du maximum et du minimum de la fonction, de la même manière qu'à l'exercice 5.10.

En 6Ø nous avons écrit MI au lieu de MIN ; on sait que cela référence la même variable. Si l'on avait écrit MIN, Basic aurait lu IF FNF(X)<M INT HEN... et aurait trouvé le mot-clé INT en position erronée. Voilà le genre de petite surprise que l'on peut avoir !

Nous avons rencontré dans les programmes de la série B qui précèdent, une bonne occasion d'utiliser un DEF FN ; c'est pour imprimer un résultat en ne conservant que deux décimales : la même expression revient plusieurs fois.

Par exemple, dans B-8, on aurait pu écrire :

```
3 DEF FNZ(X)=INT(X*1ØØ)/1ØØ
```

Les lignes 6Ø et 9Ø devenant respectivement :

```
6Ø PRINT"ERREUR";FNZ(E);"%":GOTO 3Ø
9Ø PRINT"GAGNE EN";FNZ(SC(J));"SECONDES".
```

De même, tous les calculs de temps seraient simplifiés par

```
5 DEF FNTI(X)=65535-DEEK(63Ø)
```

Exercice 6.2

Reprenez le programme de l'exercice 5.4. En principe, vous l'avez sauvé sur cassette. Vous avez une population répartie en 10 classes. Tracez l'histogramme correspondant.

Un histogramme est un diagramme formé d'autant de bâtons que de classes, chaque bâton ayant une longueur proportionnelle à l'effectif de la classe correspondante. A chaque ligne, ce n'est pas N espaces qu'il faut imprimer, mais N étoiles.

LES TROIS MODES BASSE RESOLUTION

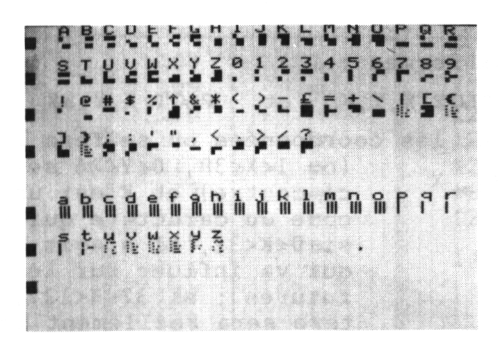
L'ORIC dispose d'autres instructions pour dessiner des courbes. Ces instructions se répartissent entre les quatre modes d'affichage que possède l'ORIC : texte, basse résolution Ø, basse résolution 1 et haute résolution. On passe d'un mode à l'autre à l'aide des instructions :

TEXT qui fait passer en mode texte;
LORES Ø qui fait passer en mode basse résolution, caractères standards;
LORES 1 qui fait passer en mode basse résolution, caractères auxiliaires;
HIRES qui fait passer en mode haute résolution étudié à la fin de ce chapitre.

Les trois modes basse résolution sont très proches et ils sont commandés par les mêmes instructions PRINT et PLOT, tandis que le mode haute résolution a tout un jeu d'instructions à part.

Les deux modes LORES ne diffèrent que par une chose : le choix du jeu de caractères utilisé. En effet, tout ordinateur doit, pour que son affichage fonctionne, posséder en mémoire les dessins de tous les caractères. Ces jeux de caractères sont normalement en ROM, mais l'ORIC les recopie en RAM à la mise sous tension. L'intérêt est que l'on peut modifier la forme des caractères; nous l'exploiterons au chapitre suivant. Une particularité de l'ORIC est qu'il dispose de deux jeux de caractères : les caractères standards (lettres majuscules et minuscules, chiffres etc...) et les caractères auxiliaires. Les caractères auxiliaires normaux ne sont pas très intéressants (caractères Télétel anglais) mais on

peut les modifier. Ils sont illustrés par la photo ci-dessous.



A la mise sous tension, on est en mode **TEXT**. Sachant que l'on peut, à l'aide des attributs afficher les caractères de l'un et l'autre jeu quelque soit le mode, nous utiliserons exclusivement le mode **TEXT** dans ce livre. Les différences entre le mode **TEXT** et les modes **LORES** concernent la gestion des attributs (cf. la section consacrée aux attributs) et la gestion des lignes et colonnes.

Rappelons que, physiquement, l'affichage de l'ORIC fait 28 lignes sur 40 colonnes. Nous numérotons les lignes de haut en bas Z, Ø, 1, ... 26. La ligne Z est réservée aux informations d'état : c'est là que s'affiche le "searching" du magnéto, le "CAPS" lorsqu'on s'est fixé en mode capitales par 'Ctrl' T etc... Nous numérotons les colonnes de gauche à droite Z, Ø, 1, 2,... 38. En mode LORES, la colonne Z est inaccessible (elle contient l'attribut de couleur du fond). Les colonnes Ø à 38 sont accessibles. En mode TEXT, la colonne Z est, comme en modes LORES, réservée à l'attribut de couleur du fond ; mais la colonne Ø est, elle aussi, réservée (à l'attribut de couleur des traits). Ceci explique qu'on ne dispose que de 38 colonnes par ligne.

PLOT

L'instruction PRINT nous a permis de dessiner sur l'écran mais elle a certains inconvénients : on a vu les difficultés qu'on a eues pour tracer les courbes sur le même graphique. En fait, ce qui est difficile c'est de se positionner à un emplacement déterminé de l'écran. On le peut à coups de CHR$(3Ø) et de mouvements de

curseur (voir tableau des CHR$ correspondants dans l'Exercice 6.1), mais l'ORIC a une meilleure instruction pour cela, PLOT.

PLOT a deux formes :

PLOT X,Y,K   et   PLOT X,Y,K$

où X et Y sont les coordonnées où se fera l'impression (ou $1\leqslant X\leqslant 38$, $0\leqslant Y\leqslant 26$ selon le schéma ci-contre) et K est un nombre, code du caractère qui sera affiché : si $0<K<31$, il s'agit d'un attribut qui va influer sur les impressions futures ; si $32<K<127$, un caractère sera réellement imprimé.

0 38 X

26 Y

Dans la 2ème forme, K$ est une chaîne de caractères qui sera imprimée à partir de la position définie par X,Y.

<u>Exemple</u> : Vérifiez que PLOT 15,15,65 et PLOT 15,15,"A" produisent le même résultat.

Le tableau suivant résume les codes caractères :

Code ASCII	*Caractère*	*Code ASCII*	*Caractère*	*Code ASCII*	*Caractère*
32	espace	53	5	75	K
33	!	54	6	76	L
34	"	55	7	77	M
35	#	56	8	78	N
36	$	57	9	79	O
37	%	58	:	80	P
38	&	59	;	81	Q
39	' (apostrophe)	60	<	82	R
40	(	61	=	83	S
41	)	62	>	84	T
42	*	63	?	85	U
43	+	64	@	86	V
44	,	65	A	87	W
	(virgule)	66	B	88	X
45	-	67	C	89	Y
46	.	68	D	90	Z
47	/	69	E	91	[
48	0	70	F	92	\
49	1	71	G	93	]
50	2	72	H	94	↑
51	3	73	I	95	£
52	4	74	J	96	©

Code ASCII	*Caractère*	*Code ASCII*	*Caractère*	*Code ASCII*	*Caractère*
97	a	108	l	119	w
98	b	109	m	120	x
99	c	110	n	121	y
100	d	111	o	122	z
101	e	112	p	123	{
102	f	113	q	124	|
103	g	114	r	125	}
104	h	115	s	126	▒
105	i	116	t		
106	j	117	u	127	DEL
107	k	118	v		

Les codes 34, 96 et 126 ne peuvent s'obtenir par "caractère" : il est donc obligatoire de les imprimer par CHR$(code).

Les codes Ø à 31 qui définissent les attributs seront étudiés plus tard.

Enfin, si l'on écrit 128+code, on obtient le même caractère mais en contraste inversé (échange des couleurs de fond et de trait) avec PLOT ou POKE, le même caractère sans inversion avec PRINT CHR$(...).

Nous sommes maintenant en mesure de tracer une courbe avec **PLOT** : nous reprenons tout simplement notre tracé de l'exponentielle (C-1A). Il faut faire une transformation des coordonnées car maintenant X est en horizontal et commence à la valeur 1. Nous appelons XX et YY les coordonnées "courbe" et X et Y les coordonnées écran :

X=1+37*XX   Y=27-YY

d'où

PROGRAMME C-1C

```
10 CLS
20 FOR XX=0 TO 1 STEP 1/38
30 X=1+INT(37*XX)
40 Y=26-INT(26*(EXP(XX)-1)/(EXP(1)-1))
50 PLOT X,Y,"*"
60 NEXT
```

Le fait de pouvoir tracer un point en fonction de ses coordonnées est très utile si la courbe est définie par des équations paramétriques.

Exercice 6.3

Tracer un cercle sur l'écran. Rappel : si XC,YC sont les coordonnées du centre et R le rayon, le cercle a pour équations paramétriques

$$\begin{cases} X=XC+R\cos T \\ Y=YC+R\sin T \end{cases} \qquad 0\leqslant T\leqslant 2PI$$

LA FONCTION SCRN

La fonction SCRN(X,Y) renvoie le code du caractère inscrit à la position X,Y. Elle renvoie 32 (espace) s'il n'y a rien.

Exemple : Après PLOT 15,15,"A" , ?SCRN(15,15) fait imprimer 65.

L'utilité de cette fonction est grande dans les programmes de jeu : vous pouvez par exemple déceler facilement si un mobile rencontre un obstacle.

ENTREES-SORTIES PAR PEEK ET POKE DANS LA MEMOIRE D'ECRAN

Une autre manière d'écrire à une position déterminée de l'écran est d'écrire directement le code voulu dans la mémoire convenable. En effet, il y a une zone mémoire assignée à l'écran, chaque position sur l'écran étant associée à une case mémoire qui conserve le code du caractère affiché à cette position.

Les adresses obéissent au schéma suivant :

```
       z      Ø      1                       37     38      X
z  | 48000  48001  48002  .............  48037  48039 ---->
Ø  | 48040  ...........                   ...    48079
1  | 48080                                          .
   |   .                                            .
   |   .                                            .
   |   .
   |   .
   |   .
   |
26 v 49080  .......                      .....  49119
Y
```

Ainsi l'adresse correspondant à la case X,Y est 48000 + 40*(Y+1)+X+1
POKE 48000+40*(Y+1)+X+1,Z est équivalent à PLOT X,Y,Z et, donc, n'a pas d'intérêt dans un cas habituel. Le seul intérêt du POKE est qu'il peut accéder à la ligne et à la colonne réservées.

De la même façon PEEK(48000+40*(Y+1)+X+1) est équivalent à SCRN(X,Y).

LES ATTRIBUTS - LES COULEURS

Lorsque le code est >32, PRINT CHR$(code) et PLOT..., code sont équivalents. Lorsque le code est inférieur, ce n'est pas le cas. Avec PRINT, on obtient une action spéciale de mise en page sur l'écran analogue à celle qu'on obtiendrait en mode direct avec la touche 'Ctrl'. Les actions de la touche 'Ctrl' ont été indiquées dans la section sur le clavier au chapitre 3. Le tableau des principaux codes de mise en page écran a été donné dans la solution de l'Exercice 6.1 au début de ce chapitre. On remarque que 'Ctrl' lettre équivaut à ?CHR$(code) si code=numéro de la lettre dans l'alphabet.

Ainsi ?CHR$(12)+"AAAA" vide l'écran et imprime AAAA. Qu'en est-il avec PLOT ? Essayez PLOT 15,15,CHR$(12)+ "AAAA".
Eh bien on imprime AAAA au milieu de l'écran, mais en clignotant. Le AAAA a été imprimé ligne 15 mais à partir de la colonne 16. Un ?SCRN(15,15) montre que la case 15, 15 a été remplie avec le code 12, qui est l'attribut du clignotement.

Peut-on avoir les attributs avec PRINT ? Oui, mais il faut, comme on l'a vu en mode direct au clavier, que le code soit précédé d'un caractère 'Escape' (CHR$(27)).
Essayez : ?" "+CHR$(27)+"LAAAA"
Vous obtenez le AAAA clignotant. L'espace en tête empêche que l'attribut vienne en tête de ligne : on verra pourquoi un peu plus loin. Le L n'est pas imprimé : c'est lui qui crée l'attribut ; on voit que ce n'est pas le code PLOT qu'il faut mettre mais la lettre dont le numéro dans l'alphabet est le code voulu. On pouvait aussi mettre :
CHR$(code + 64) : essayez ?" "+CHR$(27)+CHR$(76)+"AAAA".

Quelle est l'action des attributs ? Les attributs commandent :
- la couleur du fond, la couleur des traits des carac-

tères, le fait d'appartenir au jeu de caractères standard ou auxiliaire, le fait d'être clignotant ou non, le fait d'être en simple ou double hauteur. Les valeurs sont résumées dans le tableau ci-dessous. Les caractères avec 'Esc' ont été donnés au chapitre III.

Code des attributs

Code	*effet*	*Code*	*effet*
0	caractères noirs	15	double hauteur, auxiliaires,clignotant
1	caractères rouges		
2	caractères verts	16	fond noir (16+0)
3	caractères jaunes	17	fond rouge (16+1)
4	caractères bleus	18	fond vert (16+2)
5	caractères pourpres	19	fond jaune (16+3)
6	caractères turquoises	20	fond bleu (16+4)
7	caractères blancs	21	fond pourpre (16+5)
8	simple hauteur,standard	22	fond turquoise (16+6)
9	simple hauteur, auxiliaires	23	fond blanc (16+7)
		24	texte 60 Hz
10	double hauteur,standard	25	texte 60 Hz
11	double hauteur, auxiliaires	26	texte 50 Hz
		27	texte 50 Hz
12	simple hauteur,standard clignotant	28	graphique 60 Hz
		29	graphique 60 Hz
13	simple hauteur, auxiliaires, clignotant	30	graphique 50 Hz
		31	graphique 50 Hz
14	double hauteur,standard clignotant		

La double hauteur n'a pas grand intérêt. Les codes de 24 à 31 non plus. Notez que les valeurs marquées 60 Hz font perdre la synchronisation sur les téléviseurs européens : il ne vous reste plus qu'à faire RESET (petit bouton sur l'ORIC, ou frappe de CALL 62512) ; cela ne détruit pas le programme en mémoire.

Exercice 6.4

Imprimez BONJOUR au milieu de l'écran, clignotant en bleu sur fond pourpre.

L'exercice précédent pose le problème du *domaine de validité* des attributs. La règle est simple : un attribut est valable à partir de sa position jusqu'à la fin

de la ligne ou jusqu'à ce qu'un nouvel attribut vienne le contredire sur la ligne.

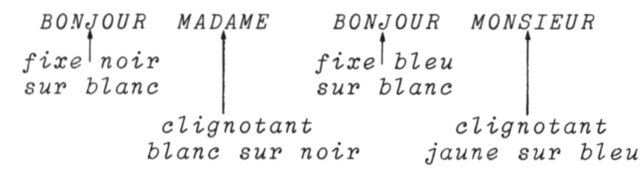
Plusieurs attributs peuvent être en vigueur en une position. Si un nouvel attribut vient contredire l'un d'eux, les autres restent valables. Ainsi, dans l'exercice 6.4, on a :

col.	14	15	16	17	23	24
	clign.	car. bleus	fond pourpre	B O N J O U R		fond blanc

En colonne 25, par exemple, on est en fond blanc, mais on est resté en caractères bleus clignotants. Vous pouvez le vérifier en venant à la fin de la ligne par mouvements de curseur et en tapant quelques caractères.

Exercice 6.5

Imprimez comme à l'exercice 3.3 :



Il y a deux manières d'obtenir du contaste inversé : changer les attributs, ou ajouter 128 au code caractère.

***Question** : Peut-on avoir, en deux positions consécutives de l'écran, des caractères de couleurs différentes ?*

- NON, car l'attribut de changement de couleur occupe une position.

Exemple : on ne peut avoir A E (tant pis pour Rimbaud..)
noir rouge

code 1
on aura au mieux A ▭ E
noir rouge

Le seul cas où l'on peut avoir des couleurs différentes consécutives sur une ligne, c'est de faire des fonds différents, sans caractères. Il suffit alors de mettre les attributs. Essayez :

```
PLOT 1Ø,1Ø,CHR$(16)+CHR$(17)+CHR$(18)+CHR$(19)+CHR$(2Ø)+
CHR$(21)+CHR$(22)+CHR$(23).
```

Exercice 6.6

Dessinez le drapeau français au milieu de l'écran.

Attributs globaux

Pourquoi y a-t-il deux colonnes réservées en début de chaque ligne ? C'est parce qu'elles contiennent les attributs de couleur de fond et de couleur des caractères standard (à la mise sous tension on a fond blanc, caractères noirs en mode TEXT ; c'est l'inverse dans les modes LORES).

Il est facile d'en changer. Il suffit, pour changer la couleur du fond de l'écran de faire POKE toutes les colonnes z, attribut de fond voulu. Essayez :

```
FOR I=48ØØØ TO 4912Ø STEP 4Ø:POKE I,18:NEXT
```

vous obtiendrez un fond vert. Ceci montre que les attributs fonctionnent aussi bien avec POKE qu'avec PLOT.

Eh bien le Basic de l'ORIC vous permet d'éviter une telle boucle. Il suffit de faire PAPER n pour définir la couleur de l'écran selon la correspondance

n	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
couleur	noir	rouge	vert	jaune	bleu	pourpre	turquoise	blanc

PAPER n équivaut à la boucle avec l'attribut n+16.

De même INK n définit la couleur des caractères avec la même correspondance. Cette fois, la valeur de n est égale à l'attribut correspondant.

Nous en avons terminé avec les modes basse résolution. La question des attributs est délicate mais c'est elle qui permet les couleurs ou des effets comme le clignotement.

Exercice 6.7

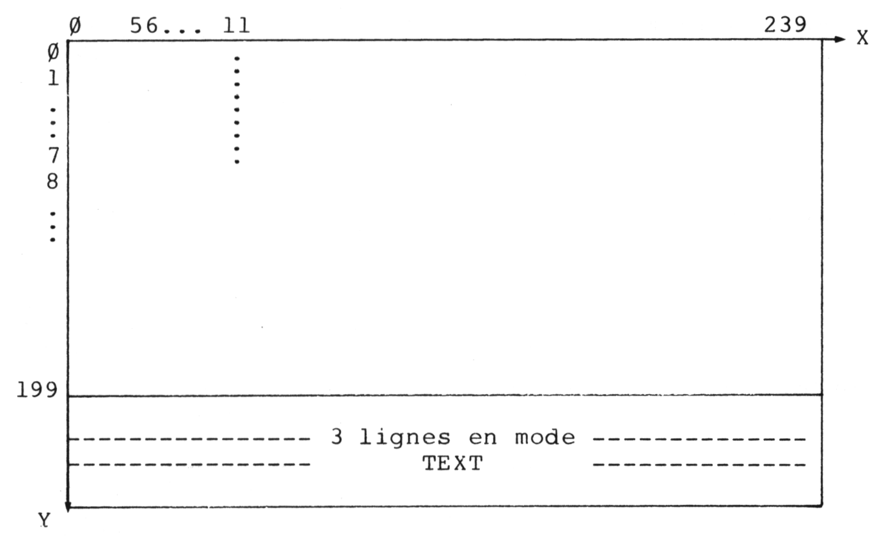
Faire un drapeau français couvrant tout l'écran.

LE PROBLEME DE LA RESOLUTION

Tous les graphiques que nous avons obtenus jusqu'à maintenant sont assez grossiers puisqu'ils sont définis

dans les mailles d'impression de l'écran : la résolution est de 28 sur 4Ø au maximum.

En fait, l'ORIC permet d'accéder aux points élémentaires de l'affichage, ce qui multiplie la résolution par 6 en horizontale et par 8 en verticale. Comme ce mode appelé HIRES (haute résolution) laisse en fait les trois dernières lignes de l'affichage en mode TEXT, ce qui vous permet de taper des commandes, la résolution finale est de 2ØØ sur 24Ø suivant le schéma :



On passe au mode haute résolution par la commande HIRES. On repasse en mode texte par TEXT. Les données de l'écran sont perdues lorsqu'on envoie une telle commande.

La fenêtre en mode TEXT est le plus souvent en noir sur blanc (elle garde les couleurs qu'on avait en mode TEXT). Les commandes PAPER et INK s'appliquent à la fenêtre en haute résolution pour définir la couleur du fond et du tracé. Lorsqu'elles sont utilisées, donc lorsqu'on ne fait pas du blanc sur noir, les colonnes élémentaires Ø à 11 deviennent inaccessibles (pour contenir les attributs voulus).

A part PAPER et INK, le mode haute résolution a un jeu de commandes spécial que nous voyons maintenant. Dans la suite X et Y dénotent des coordonnées exprimées en lignes et colonnes élémentaires :

Ø ou 12 ⩽X⩽239 ; Ø⩽Y⩽199

COMMANDES HAUTE RESOLUTION

CURSET X,Y,K : positionne le curseur en X,Y où on affiche un point de la couleur du tracé si K=1, de la couleur du fond (donc invisible) si K=Ø. Dans cette commande et les suivantes, les coordonnées n'ont pas besoin d'être entières : le système prend la partie entière.

CURMOV DX,DY,K : positionne en X+DX, Y+DY si X,Y est l'ancienne position : donc CURMOV est analogue à CURSET si ce n'est que CURSET offre un positionnement *absolu* tandis que CURMOV est *relatif* aux anciennes coordonnées. K a le même rôle que dans CURSET.

DRAW DX,DY,K : trace un trait droit de X,Y à X+DX,Y+DY. Si K=1 le tracé sera visible, si K=Ø il sera invisible (car de la couleur du fond). Le tracé est un trait continu en principe, selon la valeur donnée par l'instruction PATTERN.

PATTERN M : définit le motif des traits qui seront tracés. M est un entier de Ø à 255 c'est-à dire un octet dont les 1 définissent les pleins, et les Ø les vides.

<u>Exemple</u> : 255=11111111_2 trait continu
15=$ØØØØ1111_2$ pointillé régulier ----------
142=$1ØØØ111Ø_2$.-.-.-.

CHAR C,S,K : affiche le caractère de code ASCII C à une position telle que les coordonnées de son coin supérieur gauche soient celles du curseur. On utilise le jeu standard si S=Ø, le jeu auxiliaire si S=1. Le caractère sera visible si K=1.

CIRCLE R,K : dessine un cercle de rayon R dont le centre est à la position actuelle du curseur. Il est visible si K=1.

La commande **FILL** concerne les attributs : elle sera vue au chapitre suivant. Il reste à voir une fonction analogue à **SCRN** :

POINT (X,Y) : renvoie -1 si le point de coordonnées X,Y est allumé, Ø sinon.

Elle peut être utilisée même quand on est repassé en mode TEXT après avoir fait un dessin en HIRES.

Nous sommes maintenant en mesure de tracer une courbe en haute résolution.

Nous allons tracer 4 alternances de sinusoïde. Nous adopterons l'équation : Y=1ØØ+9Ø*SIN(8*PI*X/23Ø) d'où le programme :

PROGRAMME C-2

```
10 HIRES
20 X=0:Y=100:CURSET X,Y,1
30 FOR XX=1 TO 230
40 YY=100+90*SIN(8*PI*XX/230)
50 DX=XX-X:DY=YY-Y
60 DRAW DX,DY,1:CURSETXX,YY,1
70 X=XX:Y=YY
80 NEXT
```

Le CURSET en 6Ø est nécessaire car DRAW ne met pas le curseur à jour.

Nous sommes maintenant presque en mesure d'écrire un programme spectaculaire, le Télécran. Un petit exercice avant.

Exercice 6.8

Mettez un titre à la courbe du programme C-2.

Exercice 6.9

Transformez votre ORIC en Télécran. En fonction de la touche enfoncée, nous déplaçons le point de tracé. Si la touche 'Ctrl' n'est pas enfoncée, il y a simple déplacement. Si elle est enfoncée, il y a tracé. L'adresse mémoire 52Ø contient à tout moment l'image de la touche actuellement enfoncée. Nous proposons les assignations suivantes :

Touche	*PEEK(52Ø)*	*Mouvement*	*Touche*	*PEEK(52Ø)*	*Mouvement*
aucune	56		↓	180	↓
curseur→	188	→	espace	132	vidage écran
↑	156	↑	'Return'	175	retour en haut à gauche
←	172	←			

L'adresse mémoire 521 contient l'état des touches 'Shift', 'Ctrl' et 'Funct' selon le tableau :

Touche	*PEEK(521)*
aucune	56
Shift droit	167
Shift gauche	164
Ctrl	162
Funct	165

SAISIE AU VOL DE CARACTERES AU CLAVIER

Revenons à notre programme de jeu "devinez un nombre".

Maintenant que nous savons écrire en un endroit déterminé de l'écran, grâce à **PLOT**, nous pouvons apporter une amélioration spectaculaire à notre programme de jeu du chapitre précédent.

Quelle angoisse pour le joueur de voir les secondes s'égréner sur l'écran pendant qu'il cherche le nombre à deviner !

Pour mieux voir l'essentiel, nous partons d'une version simplifiée du programme B-5/B-8 :

PROGRAMME B-5B

```
10 DEF FNTI(X)=INT((65535-DEEK(630))/60)
15 CLS:C=1+99*RND(1):DOKE 630,65535
20 IF FNTI(X)>60 GOTO 70
30 PRINT "NB.PROPOSE ";
35 X$=STR$(FNTI(X)):X$=RIGHT$(X$,LEN(X$)-1):PLOT 35,0,X$
40 INPUT A
50 E=100*ABS(A-C)/C:IF E<1 GOTO 80
60 PRINT "ERREUR ";INT(E);"%":GOTO 20
70 PRINT"PERDU!" :GOTO 90
80 PRINT"GAGNE!"
90 INPUT"ON RECOMMENCE ";A$:IF A$="OUI" GOTO 10
```

Les simplifications que nous avons apportées sont évidentes : nous avons supprimé toute gestion des différents joueurs, simplifié le traitement des erreurs, et ramené le temps limite à 1 minute. On exploite le DEF FN vu au début du chapitre.

La fin du programme mérite examen : on demande au joueur s'il veut recommencer, et on analyse la chaîne

de caractères que forme sa réponse ; c'est très utilisé dans tous les programmes dits "interactifs".

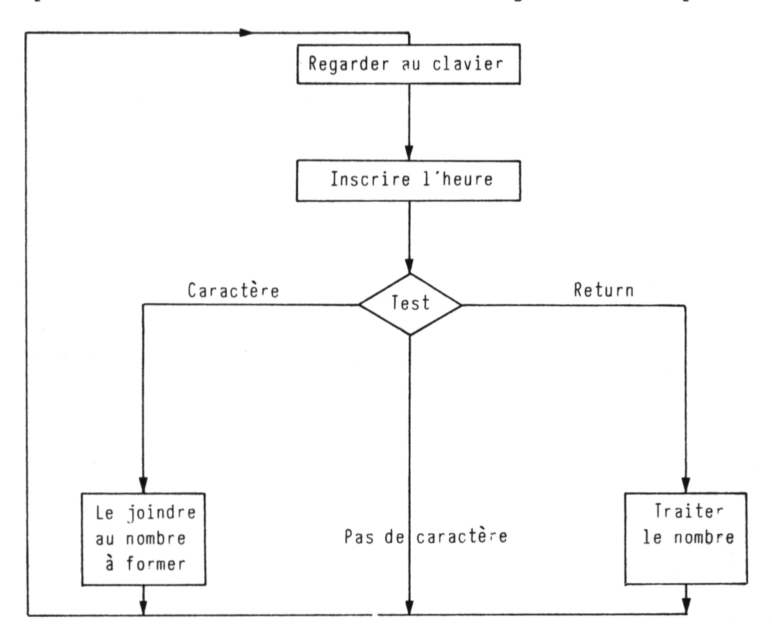
Nous avons mis le libellé "NB. PROPOSE" (ligne 3Ø) dans une instruction PRINT distincte de INPUT. C'est en effet entre les deux que se placera l'impression de l'heure. On inscrit dans la partie droite de la première ligne de l'écran le nombre de secondes écoulées. Il faut un STR$ pour transformer en chaîne de caractères. On élimine son caractère le plus à gauche car STR$ introduit un caractère parasite.

Essayons le programme ainsi obtenu.

Il ne fonctionne pas comme nous le voulons : on ne voit pas se dérouler le temps pendant que le joueur hésite à donner son nombre.

Tout vient de l'instruction INPUT. Lorsqu'une instruction INPUT s'exécute, le programme est bloqué jusqu'à l'appui sur la touche 'Return'. Pendant ce temps, rien ne peut être fait.

Ce qu'il nous faudrait, c'est une instruction qui regarde si l'on a tapé un caractère sur le clavier et qui redonne le contrôle à l'utilisateur dans tous les cas. On pourrait alors obéir à l'ordinogramme ci-après :



Eh bien l'ORIC a ce qu'il faut pour cela. On a déjà vu une solution dans l'exercice 6.9. Mais elle a un inconvénient : le PEEK(520) n'est pas le code ASCII du caractère concerné ; c'est une valeur qui dépend de l'emplacement de la touche sur le clavier. On pourrait dresser un tableau complet de la correspondance PEEK (520)-touches, mais cette solution n'est commode que si l'on s'attend à un choix très réduit de quelques touches, ce qui était le cas dans le Télécran.

L'ORIC offre deux solutions purement Basic au problème.

L'instruction GET

**GET A$** attend l'appui sur une touche et fournit le caractère tapé dans la chaîne A$.

Ceci n'est pas encore vraiment la solution à notre problème puisque GET *attend* l'appui sur une touche. Mais GET est utile pour améliorer l'interactivité du programme en évitant d'attendre en plus le 'Return'

La fonction KEY$

A$=KEY$ fournit à tout moment dans A$ le caractère correspondant à la touche enfoncée. S'il n'y a aucune touche enfoncée, on obtient la chaîne vide dans A$.

Ceci fournit la solution cherchée puisque KEY$ rend immédiatement le contrôle au programme sans attendre qu'il y ait une touche enfoncée.

En appliquant exactement l'ordinogramme précédent, on obtient :

```
      PROGRAMME  B-5C

10 DEF FNTI(X)=INT((65535-DEEK(630))/60)
15 CLS:C=1+99*RND(1):DOKE 630,65535
20 PRINT "NB.PROPOSE ?  ";
25 A$=""
30 IF FNTI(X)>60 GOTO 70
33 B$=KEY$
35 X$=STR$(FNTI(X)):X$=RIGHT$(X$,LEN(X$)-1):PLOT 35,0,X$
37 IF B$="" GOTO 30
39 IF B$=CHR$(13) GOTO 45
41 A$=A$+B$:PRINT B$;: GOTO 30
45 A=VAL(A$):PRINT
50 E=100*ABS(A-C)/C:IF E<1 GOTO 80
```

```
60 PRINT "ERREUR ";INT(E);"%":GOTO 20
70 PRINT"PERDU!" :GOTO 90
80 PRINT"GAGNE!"
90 INPUT"ON RECOMMENCE ";A$:IF A$="OUI" GOTO 10
```

Nous nous limiterons ici aux commentaires essentiels. Le changement d'ordre entre 2Ø et 3Ø est nécessaire pour que l'on passe à tout moment sur le test du temps, afin de bien assurer qu'on ne laisse qu'une minute.
On remarque, en 2Ø, que l'on imprime explicitement le point d'interrogation : GET et KEY$ ne le fournissent pas, au contraire d'INPUT. De même, avec GET et KEY$, on est obligé de contrôler la mise en page, d'où les espaces en 2Ø et 6Ø, et l'on est obligé d'imprimer les caractères à mesure qu'on les trouve, d'où le PRINT en 41.
Les lignes 25, 33 et 41 construisent progressivement la chaîne A$ à partir d'une valeur initiale "chaîne vide". A chaque fois qu'un caractère B$ est disponible, il est concaténé à la chaîne A$ déjà obtenue : A$ = A$+B$.
Notez aussi, en 39, comment le caractère 'Return' est testé ; on ne pouvait pas faire : 39 IF B$='Return', car le 'Return' termine impitoyablement la ligne. On aurait pu écrire également :

```
39 IF ASC(B$)=13...
```

Nous conseillons vivement d'adapter cette dernière version de B-5 à B-1Ø que nous avions obtenue au chapitre précédent : pratiquement, seuls les numéros d'instruction changent.

Nous voyons, en somme, que l'instruction GET et la fonction KEY$ "saisissent" les caractères un par un au clavier. Cela permet une meilleure interactivité entre l'utilisateur et la machine. Par exemple, à la fin de B-5C, on pourrait recommencer dès que l'utilisateur a tapé O de OUI, sortir dès qu'il tape N, au lieu d'attendre O U I 'Return'. Cela permet de réagir plus vite à la frappe de l'utilisateur : c'est nécessaire, en particulier lorsque, dans un jeu de bataille de chars, par exemple, l'appui sur une touche simule un tir.

Exercice 6.10

Faites la modification nécessaire à la fin du programme B-5C.

Exercice 6.11

Il est bien souvent utile d'insérer un point d'attente dans un programme : cela permet de fixer l'affichage pour que l'utilisateur en prenne connaissance. Très souvent, on décide que, pour continuer, il suffit que l'utilisateur appuie sur n'importe quelle touche. Implantez la séquence qui assure ce comportement.

GRAB ET RELEASE

Bien sûr, l'affichage en haute résolution consomme beaucoup de place mémoire. Il y a un moyen, lorsqu'on n'est pas en mode HIRES, de récupérer cette place mémoire pour Basic. Comme elle fait environ 8K, c'est appréciable sur un ORIC 16K et cela peut être utile aussi avec un 64K. Il suffit d'écrire la commande **GRAB.**

RELEASE est la commande inverse. Elle reprend à Basic la zone de mémoire de la haute résolution. Si l'on a fait GRAB pour faire tourner un programme et que l'on veut ensuite faire de la haute résolution, il faut faire RELEASE avant le HIRES.

RECAPITULATION

Ce chapitre nous a fait connaître une bonne partie des possibilités graphiques de l'ORIC. D'autres possibilités -plus délicates- seront décrites dans le chapitre suivant.

Nous savons maintenant tracer une courbe en basse ou haute résolution et manier les couleurs.

Nous avons vu en outre deux outils intéressants : DEF FN et la saisie de caractères au vol.

CHAPITRE VII

PROGRAMMES GRAPHIQUES ELABORES

LES GENERATEURS DE CARACTERES

Il nous faut maintenant expliquer un peu plus profondément comment marche l'affichage de l'ORIC. Nous allons simplifier au maximum.

L'écran haute résolution est formé d'une matrice de points (200 sur 240), chacun étant susceptible d'être allumé ou éteint. A la mise sous tension et en l'absence d'attributs, "allumé" = blanc, "éteint" = noir. Le téléviseur ou, plutôt, le processeur graphique, balaie l'image ligne élémentaire par ligne élémentaire. Pour chaque point de l'image, au moment où il le balaie, il doit être capable de savoir s'il est allumé ou éteint. Le fait d'être allumé pouvant se coder sur un bit (1 = allumé, 0 = éteint), ceci représente une information de 200x240=48000 bits. En fait, le long d'une ligne élementaire, les bits sont groupés 6 par 6 et chaque groupe entre dans un octet. Il y a donc 40 octets pour chaque ligne d'où 8000 octets.

Sur l'ORIC ces octets sont aux adresses 40960 à 48959 selon le schéma

```
       ----------------------------------------------------------> X
l.e.0 | 40960   40961   -----------                     40999
      | [][][]  [][][]\ 6bits                           [][][]
l.e.1 | 41000   41001   -----------                     41039
      | [][][]  [][][]                                    .
      | 41040                                             .
      |   .                                               .
      |   .
      |   .
      |   .
l.e   |
199   |  48920 -------------------------------------- 48959
Y     v
```

Vous pouvez vérifier ces adresses en tapant :

HIRES:POKE 4Ø96Ø,63:POKE 48959,63

Bien sûr, 8ØØØ octets c'est une taille mémoire importante. On va voir que pour afficher des textes, la taille nécessaire peut être diminuée ce qui fait gagner en mémoire et aussi en vitesse d'affichage.

Pour cela, on regroupe les points élémentaires en rectangles de 8 lignes élémentaires sur 6 colonnes élementaires, appelés les "mailles". Un caractère vient s'afficher sur l'écran dans une maille. Les mailles sont réparties en lignes et colonnes. Il faut bien distinguer ces lignes et colonnes qu'on appellera "macroscopiques" des lignes et colonnes élémentaires. L'écran est formé de 28 lignes sur 4Ø colonnes macroscopiques suivant un schéma donné au chapitre précédent.

Maintenant, on fait la remarque que les caractères que l'on a à afficher dans une maille appartiennent à un jeu restreint : alors qu'il y a 2^{48} combinaisons possibles (il y a 6x8=48 points dans chaque maille, chacun pouvant être allumé ou éteint), on n'affiche en fait que, au plus, 256 caractères possibles (y compris les contrastes inversés). On va alors coder l'information en deux fois :

Dans une première mémoire (vive obligatoirement), de nombre d'octets égal au nombre de mailles sur l'écran (donc 112Ø sur l'ORIC), on mettra le code du caractère qui doit figurer dans chaque maille. Si E est l'adresse d'origine de cette mémoire (48ØØØ), l'adresse correspondant à la maille de coordonnées X,Y est 48ØØØ+4Ø*Y+X (la formule est légèrement différente de celle de la page 97 qui tenait compte des lignes et colonnes réservées)(voir schéma page 96).

On a ensuite une seconde mémoire (qui peut être de la mémoire morte) qui va contenir la correspondance code-dessin. Cette mémoire s'appelle le générateur de caractères. Appelons G son origine (46Ø8Ø ou 471Ø4 sur l'ORIC). Il y a dans cette mémoire 8 octets pour chaque caractère. A l'adresse G+8*R+L se trouve un octet dont les 6 bits de droite sont l'image de la ligne élémentaire L qu'il faut pour dessiner le caractère de code R. L est compris entre Ø et 7. R est compris entre Ø et 127 car les motifs en contraste inversé sont déduits de leurs correspondants en contraste normal : le système utilise R AND 127 si R⩾128. En fait, seul R⩾32 donne un caractère ; en dessous, le code est considéré comme attribut.

Essayez le programme :

```
10 G=46080:R=65:REM CODE DE 'A'
20 FOR L=0 TO 7
30 PRINT PEEK(G+8*R+L):NEXT
```

Vous obtenez les impressions :

```
 8 = 00001000    soit   . . ● . . .
20 = 00010100           . ● . ● . .
34 = 00100010           ● . . . ● .
34 = 00100010           ● . . . ● .
62 = 00111110           ● ● ● ● ● .
34 = 00100010           ● . . . ● .
34 = 00100010           ● . . . ● .
34 = 00100010           ● . . . ● .
 0 = 00000000           . . . . . .
```

On voit comment cela fait dessiner un A.

L'ORIC a deux générateurs de caractères d'origines G1=46080, c'est le générateur de caractères standard et G2=47104, c'est le générateur de caractères auxiliaire.

Le jeu auxiliaire s'obtient comme on l'a vu après l'attribut 9 ou partout, en mode LORES 1. Il est formé, en normal, de petits dessins (voir photo page 93) qui correspondent au TELETEL anglais et n'ont pas grand intérêt pour nous. Mais on va changer cela.

En effet, sur l'ORIC, les deux générateurs de caractères sont en mémoire vive (en fait, il en existe une copie en mémoire morte qui est copiée dans la zone appropriée de mémoire vive à la mise sous tension ou lorsqu'on fait RESET). Cela veut dire que tous les caractères sont programmables, c'est-à-dire que leur forme est modifiable à volonté par des POKE dans les adresses voulues.

Ceci a des applications prodigieuses : on peut créer le jeu de caractères que l'on veut en modifiant certains dessins de caractères ou en les modifiant tous éventuellement. Ceci permet par exemple les lettres grecques, certains signes mathématiques, les caractères accentués du français pour un mini-traitement de textes. Cela permet de créer des caractères graphiques comme des insectes, des envahisseurs ou autres...

Nous allons par exemple créer une balle, en remplacement du caractère '*'. Pour cela nous dessinons notre

balle dans sa matrice 8x6, d'où les valeurs des lignes en binaire puis en décimal.

```
. . . . . .    ØØØØØØ  =   Ø
. ● ● ● ● .    Ø1111Ø  =  3Ø
● ● ● ● ● ●    111111  =  63
● ● ● ● ● ●    111111  =  63
● ● ● ● ● ●    111111  =  63
● ● ● ● ● ●    111111  =  63
. ● ● ● ● .    Ø1111Ø  =  3Ø
. . . . . .    ØØØØØØ  =   Ø
```

On met ensuite les données en DATA et il suffit du programme suivant :

Programme D-1A

```
1Ø PRINT "     *"
2Ø PRINT "    *  *"
3Ø G=46Ø8Ø:R=ASC("*")
4Ø FOR L=Ø TO 7:READ X
5Ø POKE G+8*R+L,X:NEXT
1ØØ DATA Ø,3Ø,63,63,63,63,3Ø,Ø
```

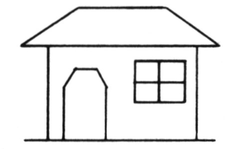
On voit que toutes les étoiles de l'écran se transforment en balles y compris celles du listing du programme. Il n'est donc peut-être pas très indiqué de transformer les caractères courants : les listings des programmes deviendraient illisibles, encore qu'on puisse toujours revenir à la normale en faisant RESET ou CALL 62512.

C'est pourquoi il est préférable de modifier plutôt le générateur de caractères auxiliaires (d'origine 471Ø4). On affiche les caractères auxiliaires en mettant des attributs 9 (retour aux caractères standard avec l'attribut 8) ou en se mettant en mode LORES 1.

Exercice 7.1

Obtenez la balle avec le jeu de caractères auxiliaires.

Nous allons maintenant dessiner une petite maison, ayant l'aspect ci-contre. Il faut d'abord faire le "répertoire" des caractères dont nous avons besoin.



Nous proposons :

(A) (B) (C) (D) (E)
(F) (G) (H) (I) (J)

d'où les données

```
A : 1,2,2,4,8,16,32       ;     B : 32,16,16,8,4,2,2,1
C : 63,0,0,0,0,0,0,0      ;     D : 0,0,0,0,0,0,0,63
E : 8 fois 32             ;     F : 8 fois 1
G : 63, 7 fois 32         ;     H : 63, 7 fois 1
I : 7 fois 32, 63         ;     J : 7 fois 1, 63
```

D'où le programme qui commence par l'impression de la maison en caractères ordinaires, d'abord. Ensuite on transforme le jeu de caractères et enfin on met les attributs 9.

PROGRAMME D-2

```
10 PRINT"           DDDDDD"
20 PRINT"          A      B"
30 PRINT"         A        B"
40 PRINT"         HCCCCCCCCG"
50 PRINT"         F ACB GGEE"
60 PRINT"         F E F GGEE"
70 PRINT"         F E F CC E"
80 PRINT"         JDIDJDDDDI"
90 G=47104
100 FOR R=65 TO 74
110 FOR L=0 TO 7
120 READ X:POKE G+8*R+L,X
130 NEXT L:NEXT R
140 FOR I=0 TO 26:PLOT 3,I,9:NEXT
150 DATA 1,2,2,4,8,16,16,32,32,16,16,8,4,2,2,1
160 DATA 63,0,0,0,0,0,0,0,0,0,0,0,0,0,0,63
170 DATA 32,32,32,32,32,32,32,32, 1,1,1,1,1,1,1,1
180 DATA 63,32,32,32,32,32,32,32,63,1,1,1,1,1,1,1
190 DATA 32,32,32,32,32,32,32,63, 1,1,1,1,1,1,1,63
```

Exercice 7.2

Faites que la maison soit imprimée plus au milieu de l'écran et qu'il n'y ait pas le Ready.

Exercice 7.3

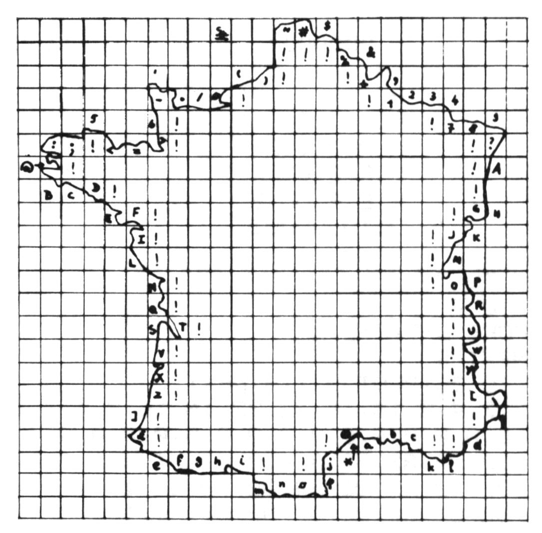
Le coin inférieur droit de la fenêtre n'est pas satisfaisant. Remédiez-y. Ajoutez une cheminée.

CARTE DE FRANCE

Pour montrer notre maîtrise des graphiques, nous allons dessiner une silhouette de la France.

La méthode est exactement la même que pour la maison à ceci près qu'il y a beaucoup plus de caractères à "programmer".

La première chose à faire est de tracer une carte de France dans une grille géométriquement semblable à la maille de l'écran. On utilise une grille de 7 sur 7,5 comme figure ci-dessous :



La figure montre aussi dans chaque carreau frontière, le caractère auquel se substituera le motif qui est dans ce carreau. On commence par espace qui reste tel quel, ! qui devient le plein puis "# etc...

La première étape sera d'imprimer la France avec ces caractères. " sera imprimé par CHR$ puisqu'on ne peut le faire autrement. On remplace le 'Copyright' qui est dans le même cas par '*' ce qui évite un 2ème CHR$. Le programme devra être tapé en mode minuscules puisqu'il y a des minuscules à faire. Attention de taper les mots-clés Basic en majuscules (avec '<u>Shift</u>').

La deuxième étape est de fournir les données de chacun des carreaux. Pour cela on examine chaque carreau superposé à une minigrille 6x8.

```
On aura :
pour !  63, huit fois
pour "  Ø,1,15,31,63,63,63,63,63
pour #  12,6Ø,62,62,62,63,63,63
-----    etc. ---
```

On aura donc une série de DATA commençant par
... DATA 63,63,63,63,63,63,63,63,Ø,1,15,31,63,63,63,63,
12,6Ø...
DATA 62,62,62,63,63,63,...

C'est trop long ! Il y a 8Ø carreaux concernés, donc 64Ø nombres. Chacun prend -avec la virgule- 2 ou 3 octets donc 2,5 en moyenne soit 16ØØ octets. Ce n'est pas tant pour la taille-mémoire que pour la frappe que nous allons essayer de simplifier.

Nous allons essayer de remplacer chacun des nombres qui viennent par un caractère. C'est possible puisque les nombres vont de Ø à 63 et on pourrait prendre comme correspondance la fonction "code-32". La valeur 2 pose une difficulté car elle correspond à " qui ne peut être mis en DATA. Heureusement, nous n'en avons pas. Sinon, il aurait fallu mettre un test. Les caractères , : ' et espace en tête doivent être fournis entre guillemets dans les DATA.

D'où le programme.

```
1000 REM
1001 REM  CARTE DE FRANCE
1002 REM
1010 DATA ________," !/?____",",\^^^___","     #>_","  PXX___","
  @PX"
1020 DATA "       X","      !/_","##'/?___","__^\XPP@",XXXY[___,88XX
XPPX
1030 DATAX_??////,PXXPPPW_,"        P\","!#'''! /",X\^_____,"   @P[__
","      R__"
1040 DATA "       @P","          _","/''''''''",PXX[____,"   R____","
  X^_"
1050 DATA "#'/?_?!)","P^______",@PPXY[__,"  #S____","'''GCA__",__^
\XXP
1060 DATA "/+!!_?'#",PPP@@@@ ,#!!      ,_G#      ,"____/'# ","?/'!''#
!"
1070 DATA _______\,____OFPX,"  @@     ",!/?////?,"_____^\X","XP@
 "
1080 DATA "?/''##! ",XXPP@@  ,_/!!###!,"'//_____","  @@@@PP","!!#''
#! "
1090 DATA PXXXP@PX,"  #''''''",??/GWWS[,XX\^^^^^,"'''''///",^\@ @PX\
1100 DATA "-!''/////",\\\XXXX\,//????__,\\^_____,"  &/___^","  !!!!#
#"
1110 DATA ^\\\XP@ ,"'?//!!##",J5J5J5J5,__?/!    ,____T   ,"______''''"
,_^XP @@@
1120 DATA "/'!      ","___?/'  ",______?/,____^^^^,"______/''",\XXPXX
XX
1130 DATA ?/-!    ,^T@@    ,__/##!  ,_______9,_______,XXXX\\\(,J5J
5J5J5
```

```
1200 G=47104:FOR R=33 TO 113
1210 READ A$:FOR L=0 TO 7
1220 A=ASC(MID$(A$,L+1,1))-32
1230 POKE G+8*R+L,A:NEXT:NEXT
1300 CLS:SP$="":PL$=""
1310 FOR I=1 TO 40:SP$=SP$+" ":PL$=PL$+"!":NEXT
1320 PRINT:PRINT:PRINT:PRINT
1330 PRINTLEFT$(SP$,21);CHR$(34);"#$"
1340 PRINTLEFT$(SP$,21);"!!!%&"
1350 PRINTLEFT$(SP$,15);"'   ()!!!!+,"
1360 PRINTLEFT$(SP$,15);"-./0!!!!!!!1234"
1370 PRINTLEFT$(SP$,12);"5  6!!!!!!!!!!!!!789"
1380 PRINT"         :;!<=>";LEFT$(PL$,15);"?"
1390 PRINT"         @";LEFT$(PL$,20);"A"
1400 PRINT"         BCD";LEFT$(PL$,18)
1410 PRINT"            EF";LEFT$(PL$,15);"GH"
1420 PRINTLEFT$(SP$,14);"I";LEFT$(PL$,14);"JK"
1430 PRINTLEFT$(SP$,14);"L";LEFT$(PL$,14);"M"
1440 PRINTLEFT$(SP$,15);"N";LEFT$(PL$,13);"OP"
1450 PRINTLEFT$(SP$,15);"Q";LEFT$(PL$,14);"R"
1460 PRINTLEFT$(SP$,15);"ST";LEFT$(PL$,13);"U"
1470 PRINTLEFT$(SP$,15);"V";LEFT$(PL$,14);"W"
1480 PRINTLEFT$(SP$,15);"X";LEFT$(PL$,14);"Y"
1490 PRINTLEFT$(SP$,15);"Z";LEFT$(PL$,14);"[\"
1500 PRINTLEFT$(SP$,14);"]";LEFT$(PL$,16);"^"
1510 PRINTLEFT$(SP$,14);"_!!!!!!!!!!*abc!!d"
1520 PRINTLEFT$(SP$,15);"efghi!!!j   kl"
1530 PRINTLEFT$(SP$,20);"mnop"
1600 FOR I=1 TO 26:PLOT 3,I,9:NEXT
1610 GOTO1610
```

Dans le listing ci-dessus, à cause de l'imprimante utilisée, le signe £ ("livre sterling") apparaît comme un "souligné" ('_'), par exemple, voir la ligne 1Ø1Ø.

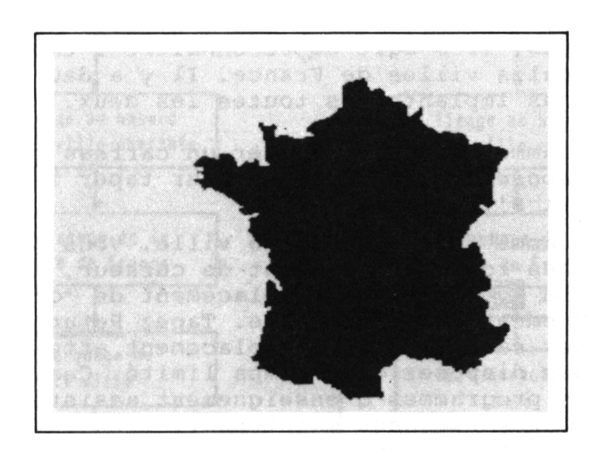
De 1Ø1Ø à 113Ø on a les DATA voulues. On a ajouté le dessin d'un damier en remplacement de 'q' car cela nous servira ultérieurement.

De 12ØØ à 123Ø, on trouve la conversion des données et leur incorporation en mémoire.

De 13ØØ à 153Ø, on trouve l'impression de la France avec les caractères du jeu standard. Remarquez comment on imprime des espaces ou des ! multiples. La France n'apparaît telle que nous la connaissons qu'après exécution de 16ØØ qui fait passer au jeu auxiliaire.

Le remplissage du générateur de caractères est long mais une fois qu'il est fait, on peut réobtenir la France en faisant seulement RUN 13ØØ. La photo de la page suivante montre le résultat obtenu. Les caractères qui apparaissent sur la ligne du haut sont dûs à une petite erreur de conception de l'ORIC : le 2ème générateur de caractères et la mémoire d'écran se recouvrent à partir

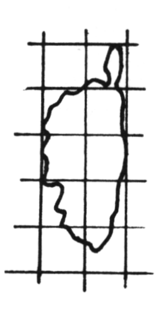
de l'adresse 48000 (code caractère 112, c'est-à-dire 'p'). Tant qu'on ne redéfinit pas les caractères jusque-là, cela va ; sinon, les premières lignes de l'écran sont troublées.



Nous demandons aux lecteurs Corses de bien vouloir nous excuser de ne pas avoir fait figurer leur belle Ile sur notre carte, ainsi d'ailleurs qu'aux habitants des îles côtières qui n'apparaissent pas. Pour les dédommager, nous proposons l'exercice suivant :

Exercice 7.4

Cartographiez la Corse grâce à la grille ci-contre.



Exercice 7.4 Bis

Pour les lecteurs francophones, faites de même une carte de la Belgique, ou de la Suisse, ou du Québec...

QUIZ GEOGRAPHIQUE - LES SOUS-PROGRAMMES

Maintenant que nous disposons d'une carte de France, nous allons pouvoir l'utiliser pour jouer à un jeu utile. Il va nous permettre de réviser nos connaissances géographiques. Il s'agit de reconnaître l'emplacement des principales villes de France. Il y a deux formes de jeu, que nous implanterons toutes les deux.

A. Le programme marque en damier un carreau sur l'écran ; vous disposez de 2Ø secondes pour taper le nom de la ville qui s'y trouve.

B. Le programme donne un nom de ville. Vous devez, à l'aide des touches mouvement de curseur, amener le caractère ▒ (damier, remplacement de 'q') sur l'emplacement de cette ville. Tapez Return quand vous êtes satisfait de l'emplacement atteint. Bien sûr, vous disposez d'un temps limité. Ceci est la base des programmes d'enseignement assisté par ordinateur, très répandus maintenant. Bien entendu, on peut les perfectionner en examinant les erreurs faites, en affichant des indications pour vous aider et en comptabilisant les résultats.

Nous avons vu précédemment tous les éléments qui permettent de le faire ; nous nous bornerons ici aux solutions les plus simples.

Le programme obéit à l'ordinogramme général ci-contre.

Cet ordinogramme est simplifié. Ce qui apparaît fondamentalement, c'est que, en deux endroits, il faut exécuter la même opération : afficher la carte de France. Le programme de la carte est bien trop long pour le répéter en deux endroits.

Choix de la forme
A ou B

A
Oui
Non

Tirage au hasard
de la ville cherchée

Affichage de la
carte de France

Marquage de
la ville

Lecture de la réponse

Bonne
Non
Oui

Temps
écoulé
Non

X
X

Tirage au hasard
de la ville cherchée

Affichage de la
carte de France

Lit caractère
tapé

Test
Vide
Curseur

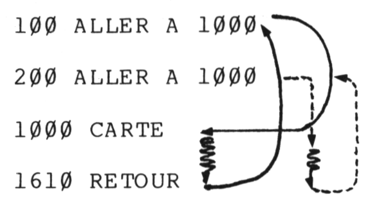
Effectue le
mouvement

Return

Analyse de
la réponse

X

Ce qui serait bon, c'est de disposer du mécanisme suivant. (cf figure ci-dessous).



Le programme à répéter est écrit une seule fois, à partir de la ligne 1000 par exemple.

A chaque fois qu'on doit l'exécuter (ici en 100 et 200) une instruction qui ressemble à un GOTO fait sauter à 1000, où la séquence est exécutée.

La séquence se termine par une instruction qui veut dire "Retournez d'où vous venez". Etant venu de la ligne 1ØØ, retourner juste en-dessous de 1ØØ (parcours en trait plein). Etant venu de 2ØØ, retourner juste en-dessous de 2ØØ (parcours pointillé).

En somme, c'est un mécanisme de saut qui se "souvient" d'où il vient. Ce mécanisme existe en Basic.

Le programme utilisé plusieurs fois s'appelle un sous-programme. L'instruction de saut vers le sous-programme est l'instruction d'appel *GOSUB*: 1ØØ GOSUB 1ØØØ.

L'instruction de retour s'écrit tout simplement *RETURN* (les lettres R E T U R N, pas la touche 'Return').

Le programme a donc la structure :

```
:
1ØØ GOSUB 1ØØØ     }
:                  }  programme principal
2ØØ GOSUB 1ØØØ     }
:
1ØØØ  :            }  sous-programme (ici de tracé
      :            }  de la carte de France).
161Ø RETURN        }
```

Cette structure est extrêmement importante et puissante. Un programme peut appeler différents sous-programmes. Chaque sous-programme peut en appeler d'autres. Dans tous les cas, l'ORIC s'y retrouvera pour effectuer les retours.

Cette simplification de notre travail étant acquise, nous avons besoin d'une liste de villes avec leur nom et, pour chacune, ses coordonnées. X (de 1 à 38) et Y (de 1 à 26) qui la situent sur l'écran. Bien sûr, ces coordonnées seront approximatives, comme le dessin de la carte.

Nous nous bornerons ici aux 11 villes suivantes (cela évite les instruction DIM, et rien n'empêche d'en ajouter ultérieurement).

Villes	*X*	*Y*	*N*
Paris	23	9	Ø
Marseille	28	22	1
Lyon	27	16	2
Toulouse	21	21	3
Nice	32	21	5
Bordeaux	18	18	5

Villes	*X*	*Y*	*N*
Nantes	16	13	6
Strasbourg	31	1Ø	7
St-Etienne	26	17	8
Le Havre	19	7	9
Lille	24	5	1Ø

Les préliminaires du programme sont alors simples. Le tableau précédent est mis sous forme de DATA (lignes 1Ø, 15, 2Ø) qui sont ensuite lues (lignes 3Ø et 4Ø). Ensuite, c'est le choix du mode (ligne 5Ø et 6Ø). (Pour une utilisation réelle du programme, il faudrait imprimer beaucoup plus d'explications que nous ne le faisons).

Avant de taper ce programme, chargez votre programme carte de France et ajoutez une ligne 1ØØØ REM. Changez la ligne 161Ø en 161Ø RETURN. Ajoutez aussi un 125Ø RETURN: cela sépare en un autre sous-programme le remplissage du générateur de caractères qui n'a pas à être répété.

```
        PROGRAMME D-3  QUIZ GEOGRAPHIQUE

10 DATA PARIS,MARSEILLE,LYON,TOULOUSE,NICE,BORDEAUX,
                                                  NANTES
15 DATA STRASBOURG,SAINT ETIENNE,LE HAVRE,LILLE
20 DATA 23,9,28,22,27,16,21,21,32,21,18,18,16,13,31,10,
                                          26,17,19,7,2
4,5
30 FOR V=0 TO 10:READ NOM$(V):NEXT
40 FOR V=0 TO 10:READ II(V),JJ(V):NEXT
45 CLS:PRINT"UN PEU DE PATIENCE SVP":GOSUB 1000
50 CLS:INPUT"A OU B ";X$
55 V=INT(11*RND(1))
60 IF X$<>"A" GOTO 500
```

En fait, le tirage de la ville voulue est commun et en 55 :

```
55 V=INT(11*RND(1)):
```

Les initialisations du temps se font par un simple DOKE 63Ø,65535 en 72 et 51Ø.

Jeu ***A***

On se trouve après 7Ø :

```
70 PRINT"JEU A:VOUS TAPEZ LE NOM DE LA VILLE   MARQUEE
                                          ";CHR$(126);
" PUIS Return"
```

```
72 GOSUB 5000:DOKE 630,65535
75 X=II(V):Y=JJ(V)
80 GOSUB 1300:C$="":PLOT X,Y,113
85 IF 65535-DEEK(630)>2400 GOTO 170
95 B$=KEY$:IF B$="" GOTO 85
100 IF B$=CHR$(13) GOTO 150
105 PRINT B$;
110 C$=C$+B$:GOTO 85
150 IF C$=NOM$(V) GOTO 180
160 CLS:PRINT"NON":GOSUB 5000:GOTO 75
170 CLS:PRINT"PERDU":GOTO 190
180 CLS:PRINT"GAGNE"
190 INPUT"ON RECOMMENCE ";X$
195 IF X$="OUI" GOTO 50
200 END
```

Le programme est bâti avec des matériaux déjà vus, notamment la construction caractère par caractère du nom C$ de la ville proposée par l'élève (instructions 8Ø à 11Ø).
En 72 et en 16Ø, délais pour fixer l'affichage.
En 8Ø, on affiche la carte par GOSUB 13ØØ (le générateur de caractères a été rempli une bonne fois par GOSUB 1ØØØ en 45). 113 est le code du damier pour marquer la ville ; on l'inscrit aux coordonnées X et Y de la ville, calculées en 75.

Les délais sont faits par boucle, car WAIT perturberait les calculs de temps. Là encore, on utilise un sous-programme en 5ØØØ.
Le reste du programme ne devrait pas faire de difficulté ; nous passons donc au jeu B.

Jeu ***B***

Nous sommes cette fois après la ligne 5ØØ

```
500 PRINT"JEU B: AVEC LES TOUCHES CURSEUR, VOUS AMENEZ
                                              ";CHR$(126);
502 PRINT" SUR LA VILLE VOULUE"
505 PRINT"  ***  ";NOM$(V);"  ***":GOSUB 5000
510 X=11:Y=4:DOKE 630,65535
515 GOSUB 1300
520 C=SCRN(X,Y):PLOT X,Y,113
525 IF 65535-DEEK(630)>8400 GOTO 170
530 B$=KEY$:IF B$="" GOTO 525
535 IF B$=CHR$(13) GOTO 570
537 PLOT X,Y,C
540 IF B$=CHR$(9) THEN X=X+1
545 IF B$=CHR$(8) THEN X=X-1
550 IF B$=CHR$(10) THEN Y=Y+1
```

```
555 IF B$=CHR$(11) THEN Y=Y-1
560 GOTO 520
570 IF X=II(V) AND Y=JJ(V) GOTO 180
575 CLS:PRINT"NON":GOSUB 5000:GOTO 515
```

X et Y initialisés à 4 et 11, indiquent l'emplacement du curseur gris (code écran 113).
Là encore, pour des raisons d'interactivité, on utilise KEY$ pour examiner la touche enfoncée. Selon la touche utilisée, le mouvement est réalisé, c'est-à-dire que X ou Y sont modifiés. La touche 'Return' est testée par CHR$ puisqu'on ne peut la mettre entre guillemets.

Le mouvement, ligne 537 et 520, est réalisé, replaçant l'ancien contenu C en X,Y. Le curseur gris est alors déplacé à la nouvelle position.

Lorsque l'utilisateur tape 'Return', le X et le Y atteints sont alors l'emplacement proposé. Celui-ci est comparé aux données des tableaux II et JJ.

L'ordre des instructions, en 510, 515 est tel que lorsque la réponse est NON une autre tentative est possible. On reprend alors à la position proposée à l'essai précédent, ce qui permet d'atteindre le but par des faibles corrections successives.
Pour l'ensemble des essais, le temps est limité à 3 mn (contre 30 s dans le jeu A).

Le sous-programme en 5000 crée un délai.

```
5000 FOR R=1 TO 1000:NEXT:RETURN
```

RECAPITULATION

Ce programme, qui est l'archétype des programmes utilisés en enseignement assisté par ordinateur, n'utilise que les techniques simples qui ont été progressivement introduites au cours du jeu "devinez un nombre" et pour la programmation du dessin d'une maison.

La seule technique nouvelle introduite ici - et elle est très importante - est celle des **sous-programmes.**

Avec un simple GOSUB, le tracé de la carte de France est effectué chaque fois que nécessaire dans le programme.

La fonction KEY$ apporte également une bonne interactivité dans le dialogue homme machine, essentielle dans cette application.

Nous ne proposons pas ici d'exercice sur ce programme, construit pas à pas, mais nous ne saurions trop encourager le lecteur à s'exercer à y apporter toutes les modifications qu'il jugerait bon d'essayer. Vous pouvez notamment vous inspirer des dernières versions du programme B pour gérer les "scores" obtenus par différents élèves. Vous pouvez aussi changer les délais et ajouter des villes (Dans ce cas, attention à DIM...)

Remarque : Le sous-programme carte de France a été écrit à partir de 1ØØØ, de façon à en permettre une utilisation dans plusieurs programmes. Vous le stockez sur cassette, et lorsque vous souhaitez l'incorporer à un programme à écrire, vous le chargez en premier. Puis vous écrivez votre programme de Ø à 999.

Cette façon de procéder, qui nous a évité la recopie fastidieuse d'une partie de programme, est généralisable. Mais attention ! Vous ne pouvez l'utiliser qu'une fois par programme.

Maintenant que nous avons des affichages très élaborés, il nous reste à donner un peu de mouvement.

NOTIONS SUR LES DESSINS ANIMES

Nous nous bornerons ici aux premières notions permettant d'animer l'affichage de l'ORIC.

La première méthode qui vient à l'esprit est la méthode classique utilisée au cinéma. La scène est décomposée en un grand nombre de dessins qui sont projetés successivement.

Avec l'ORIC, qui est pourtant un des micro-ordinateurs personnels les plus rapides, se pose le problème du temps de remplissage de l'écran pour obtenir une image. D'autant plus que, pour ne pas avoir trop de scintillement, il faut afficher au moins 15 images seconde. Ceci nécessite le recours au langage machine qui est étudié dans un autre ouvrage de cette collection.

Pour pouvoir utiliser Basic, il faut passer d'une image à la suivante en ne changeant qu'une très petite

partie de l'image. On peut utiliser, au choix, des PRINT ou, mieux (plus rapide), des POKE dans la mémoire d'écran ou des PLOT.

MOUVEMENTS D'UNE BALLE

A titre d'exemple d'animation, sans modification importante d'une image à la suivante, nous allons faire se déplacer une balle sur l'écran. Cela conduira à un embryon de jeu de tennis. La balle, nous l'avons par modification du générateur de caractères au début de ce chapitre.

Pour déplacer la balle de gauche à droite, il faut :

- créer la balle. On implante le programme D-1A sous forme de sous-programme à partir de 5ØØ et on fait 1Ø GOSUB 5ØØ.
- imprimer la balle
  2Ø X=... :Y=... :PLOT X,Y,"*"
- revenir sur la balle et la remplacer par un blanc, puis imprimer une nouvelle balle 1 cran plus loin.
  3Ø PLOT X,Y,32:X=X+1:PLOT X,Y,"*"
- boucler sur l'instruction 3Ø : 4Ø GOTO 3Ø

Essayez le programme formé par les trois instructions, 1Ø, 2Ø, 3Ø ci-dessus.

Il y a plusieurs défauts :

- il faudrait vider l'écran et se placer sur la ligne centrale (13) ;
- lorsque la balle a parcouru tout l'écran de gauche à droite, on a un message d'erreur ;
- il n'y a pas de réglage de vitesse.

La première objection se résout facilement par le choix Y=13.

Pour la troisième, il faut introduire un délai avant de faire le GOTO en 3Ø, par exemple à l'aide de l'instruction WAIT :
4Ø WAIT D:GOTO 3Ø

D sert à régler le délai dans la vitesse de déplacement.

Pour la deuxième objection, on peut décider le mouvement suivant : aller de gauche à droite, puis, quand la

balle arrive à l'extrémité de la ligne, aller de droite à gauche etc.
Pour cela, il faut tester X. Le sens est inversé quand X = 38 ou 1.

Pour que la balle se déplace à gauche, il faut écrire :

```
50 :PLOT X,Y,32:X=X-1:PLOT X,Y,"*"
```

D'où le programme :

PROGRAMME D-4

```
10 GOSUB500:CLS
20 X=1:Y=13:PLOT X,Y,"*":D=6
30 PLOT X,Y,32:X=X+1:PLOT X,Y,"*"
40 WAIT D:IF X<38 GOTO 30
50 PLOT X,Y,32:X=X-1:PLOT X,Y,"*"
60 WAIT D:IF X>1 GOTO 50
70 GOTO 30
500 G=46080:R=ASC("*")
510 FOR L=0 TO 7:READ X
520 POKE G+8*R+L,X:NEXT
530 RETURN
540 DATA 0,30,63,63,63,63,30,0
```

Avec D = 6, vous réalisez un véritable hypnotiseur.
Avec D < 3, la balle va assez vite pour que l'on croie voir un sillage. Avec D > 10, les saccades du mouvement sont apparentes.

Exercice 7.5

Faire osciller la valeur du délai entre 1 et 10. (Le délai varie de 1 à chaque aller et retour).

Exercice 7.6

Faire osciller la balle du haut en bas de l'écran.

Exercice 7.7

Faire aller la balle en diagonale (du coin gauche à la dernière ligne position 26 et retour).

TENNIS

Nous sommes prêts maintenant à jouer au tennis. En fait, notre jeu sera rudimentaire. Partons de la balle qui oscille de gauche à droite, nous disposons un joueur à droite et un joueur à gauche. Le joueur de droite ne doit pas laisser la balle passer la position 38 sur l'écran. Pour manifester qu'il rattrape la balle, il doit appuyer sur une touche. Nous adopterons la touche /. Pour le joueur de gauche ce sera la touche Z. Le programme reconnaîtra la touche grâce à un KEY$.

Par ailleurs, nous imposons aux joueurs de ne pas appuyer trop tôt sur leur touche : si le joueur de droite appuie sur / avant que la balle ne soit en colonne 34, ou si le joueur de gauche appuie sur Z avant qu'elle ne soit en colonne 5, il concède un point à son adversaire.

A chaque point, le score est affiché. Le service sera effectué en appuyant sur une touche quelconque. Lors du service, la balle part du milieu de l'écran. Le sens du mouvement est tiré au sort.

Enfin on a implanté le mouvement de la balle dans un sous-programme : 4ØØ.

D'où le programme D-5 dont il y aurait lieu de perfectionner la présentation pour obtenir un jeu effectif. En particulier, le fait que les colonnes z et Ø soient blanches est trompeur pour le joueur de gauche.

PROGRAMME D-5
MINI-TENNIS

```
10 GOSUB500:SG=0:SD=0:D=3
15 CLS:PRINTSG;SPC(30);SD:K=19:GET A$
20 IF RND(1)>.5 GOTO 45
25 WAIT D:K=K+1:IF K=39 THEN SG=SG+1:GOTO 15
30 GOSUB 400:A$=KEY$:IF A$<>"/" GOTO 25
35 IF K>=34 GOTO 45
40 SG=SG+1:GOTO 15
45 WAIT D:K=K-1:IF K=0 THEN SD=SD+1:GOTO 15
50 GOSUB 400:A$=KEY$:IF A$<>"Z" GOTO 45
55 IF K<=5 GOTO 25
60 SD=SD+1:GOTO 15
400 PLOT X,13,32:X=K:PLOT X,13,"*":RETURN
500 G=46080:R=ASC("*")
510 FOR L=0 TO 7:READ X
520 POKE G+8*R+L,X:NEXT
530 RETURN
540 DATA 0,30,63,63,63,63,30,0
```

LA COULEUR EN HAUTE RESOLUTION - LA COMMANDE FILL

Il nous reste à voir une dernière instruction graphique de l'ORIC, très puissante mais assez délicate, **FILL.**

Revenons d'abord au schéma de l'écran haute résolution en tête de ce chapitre. Si l'on met par POKE un octet à une adresse de 4Ø96Ø à 48959 on doit avoir un affichage sur l'écran : 6 points élémentaires correspondant à la case mémoire vont se trouver allumés ou éteints selon la valeur (qui, elle, est sur 8 bits) écrite dans la mémoire.

Toute la difficulté est dans la correspondance entre le motif binaire d'allumage (6 bits) et la valeur à mettre (8 bits). Voici la marche à suivre :

1- déterminer d'abord le motif d'allumage. On procède exactement comme nous avons fait pour une ligne élémentaire de caractère programmable.

Exemple : supposons qu'on veuille un pointillé comme

on a ØØ11Ø1 en binaire, donc la valeur 13.

2- le problème est que l'ORIC considère les valeurs inférieures ou égales à 31 comme des attributs et non des motifs d'allumage.

La règle est la suivante : *si la valeur obtenue est ⩽31, on ajoute 64*. Dans notre exemple, on obtiendrait 77. Essayez : HIRES:POKE 4Ø96Ø,77. Vous obtenez bien

Si la valeur est supérieure ou égale à 32, on peut au choix la prendre telle quelle ou ajouter 64, le résultat est le même.

Que font les octets dont le bit 7 est à 1 ? Eh bien le bit 7 (qui revient à ajouter 128) a pour effet *d'inverser* le motif dessiné.

Exemple : POKE 4Ø96Ø,2Ø5 donne le pointillé

Maintenant, si k⩽31, 128+k donne le même attribut.

En résumé

Ø : 31	attributs	128 : 159	attributs
32 : 63	dessins	16Ø : 191	dessins inverses de 32...63

64 : 95	dessins	192 : 223	dessins inverses
96 : 127	mêmes dessins que 32..63	224 : 255	dessins inverses

Etant donné un motif, il y a trois manières de l'obtenir. Soit k la valeur 6 bits correspondante. 0⩽k⩽63.

Si k⩽31, on a déjà ℓ1=k+64. Soit k'=63-k ; c'est la valeur du dessin complémentaire. k' est >31 donc le dessin complémentaire s'obtient par k' et k'+64. Par suite le dessin primitif s'obtient par k=128+k'=191-k et k=128+k'+64=255-k.

Exercice 7.8

Donnez les valeurs si k⩾32.

Pour les valeurs ℓ⩽31, les attributs jouent le même rôle qu'on a vu en basse résolution, résumé par le tableau de la page 98. La propriété fondamentale qui demeure est qu'un attribut est valable depuis sa position jusqu'à la fin de la ligne élémentaire où il se trouve, à moins que, plus à droite, sur cette ligne élémentaire on ne rencontre un autre attribut qui le contredise : c'est alors lui qui devient valable jusqu'à la fin de la ligne élémentaire.

Lorsqu'on met un attribut de couleur de fond, on obtient toute la fin de la ligne élémentaire concernée de la couleur indiquée.

Exemple : POKE 40960,20 rend bleue toute la 1ère ligne. Si l'on fait ensuite POKE 40961,17, la ligne devient rouge, sauf le premier segment qui reste bleu.

Il y a une constatation importante à faire : on ne peut pas, en horizontal, faire varier les couleurs sur une distance inférieure à un segment (c'est-à-dire 6 points élémentaires). Au contraire, en vertical, la couleur peut changer d'une ligne élémentaire à l'autre.

Autrement dit, en mode haute résolution, la résolution pour les tracés en deux couleurs (fond et courbe) est de 199 sur 240, tandis que la résolution pour les dessins multicolores n'est que de 199 sur 40.

L'instruction FILL

On pourrait faire des dessins haute résolution par POKE en 4Ø96Ø et suivantes : on a vu comment éclairer les points voulus. Mais -on l'a vu au chapitre précédent- le Basic de l'ORIC a les instructions CURSET, DRAW etc... qui sont beaucoup plus simples d'emploi.

Eh bien lorsqu'il s'agit de placer des attributs ou des motifs, le Basic offre la commande FILL. Elle est de la forme :

```
FILL NLE,NS,M
```

Ceci remplit du même motif M,NS segments sur NLE lignes élémentaires. (1≤NLE≤2ØØ ; 1≤NS≤4Ø Ø≤M≤255). Si M est un attribut, il suffit que NS=1.

Oui, mais où commence le remplissage ? A la position X,Y définie par le dernier CURSET.

<u>Exemple</u> : faire un fond jaune à la moitié droite de <u>l'écran</u> :

```
CURSET 12Ø,Ø,Ø :FILL 2ØØ,1,19
```

Exercice 7.9

Faire un fond jaune sur tout l'écran.

Exercice 7.10

Faire le drapeau français.

Exercice 7.11

Couvrir la moitié haute de l'écran avec des rayures verticales très fines et la moitié basse avec des rayures moins fines.

Il faut noter que, avant un FILL, CURSET X,..., CURSET X+1,..., CURSET X+2,... ...CURSET X+5,... sont équivalents puisque FILL ne peut porter que sur des segments entiers. Quelque soit X, le FILL commence au segment n° INT(X/6).

En outre, FILL ne met pas le curseur à jour. Tous les FILL doivent être précédés d'un CURSET.

Nous allons appliquer ceci pour dessiner un disque de couleur. Pour cela, on va faire un FILL à partir de

chaque segment gauche du disque donné par l'équation XX=2Ø-INT(19*SQR(1-(Y-1ØØ)↑2/1ØØØØ)) et on refait un FILL du fond noir à partir du segment de droite donné par XX=2Ø+INT(19*SQR(1-(Y-1ØØ)↑2/1ØØØØ)).

Le 19 (multiplicateur de SQR) a peut-être à être rectifié en fonction de votre téléviseur pour que le disque soit bien rond. Sur le nôtre, nous avons été amenés à la formule ...INT(Ø.5+14*SQR...)

PROGRAMME D-6

```
10 HIRES
20 FOR Y=1 TO 199
30 D=INT(0.5+14*SQR(1-(Y-100)^2/10000))
40 X1=6*(20-D):X2=6*(20+D)
50 CURSETX1,Y,0:FILL1,1,20
60 CURSETX2,Y,0:FILL1,1,16
70 NEXT
```

Exercice 7.12

Faites que le disque soit clignotant (toujours bleu sur noir).

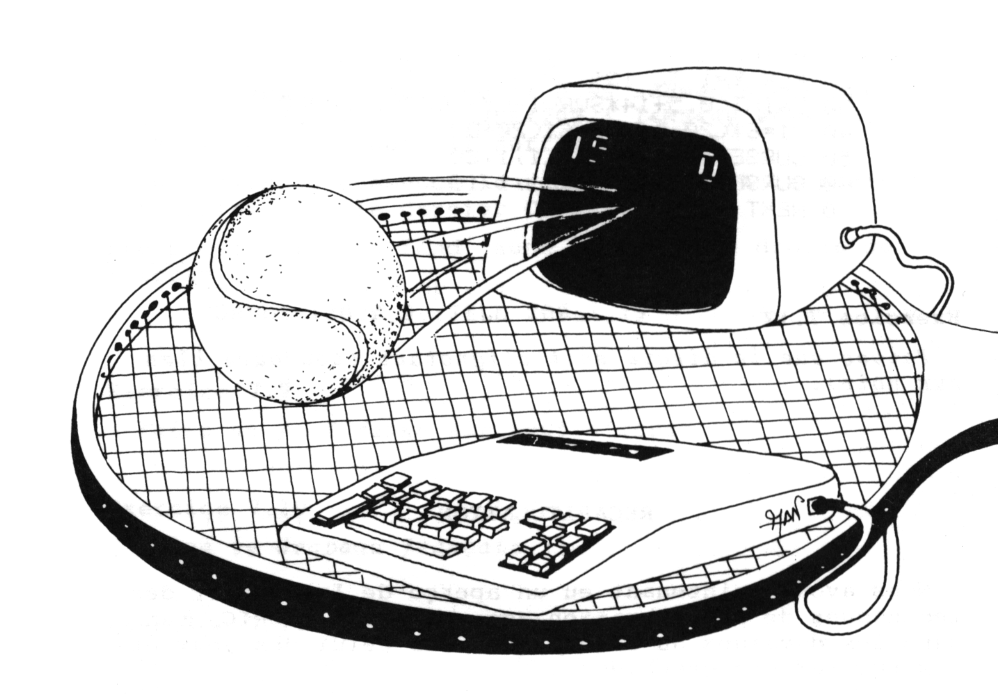
RECAPITULATION

Nous avons maintenant eu un aperçu de la plupart des techniques de programmation accessibles à l'ORIC, tant dans les domaines du calcul que dans celui des jeux et des traitements graphiques.

A propos des dessins animés, nous avons vu que le Basic est un peu lent pour certaines applications.

Dans ce cas, il y a un recours c'est le langage machine du microprocesseur de l'ORIC.

Les techniques correspondantes seront abordées dans un autre ouvrage de cette collection.



CHAPITRE VIII

EFFETS SONORES

Voilà un chapitre qui va faire du bruit ! En effet, l'ORIC possède un petit haut-parleur avec lequel il est capable de produire des sons variés.

Il y a d'abord quatre instructions qui font chacune un bruit prédéfini, très utile dans les jeux :

EXPLODE : bruit d'explosion ;
PING : bruit de triangle ("ding" très aigu) ;
SHOOT : bruit de coup de feu ;
et ZAP : bruit de canon laser. Gare ! si les Extra-terrestres se mettent à tirer !

Il faut noter que la combinaison de touches 'Ctrl' G, ou encore PRINT CHR$(7) produit le même son que PING. On peut même le dissimuler dans une chaîne de caractères :

```
A$="XXX"+CHR$(7)+"YYY":PRINT A$
```

Ces sons prédéfinis ne sont que des combinaisons des instructions sonores de base SOUND, MUSIC et PLAY. Ils mettent en jeu trois voix indépendantes, chacune pouvant être mélangée à une voix de bruit.

SOUND est de la forme :

SOUND NV,P,V

où • NV est le numéro de voix 1, 2 ou 3
si NV=4,5 ou 6, la voix (resp. 1,2 ou 3) est mélangée à du bruit.

- P est proportionnelle à la période du son. La fréquence est donnée approximativement par F= 62ØØØ/P. Cette formule n'a pas d'importance puisque les notes peuvent être obtenues directement par MUSIC.
- V est le volume de 1 (le plus faible) à 15 (trop fort!).
  Prenez une valeur moyenne de 5 à 8. Si V=Ø, on n'a pas de son immédiatement : on le prépare pour exécution ultérieure par PLAY.

Exemple : SOUND 1,14Ø,1Ø donne le la du violon

MUSIC est de la forme :

MUSIC NV,OC,NO,V

où
- NV est le numéro de voix 1, 2 ou 3 (pas de 4, 5 ou 6 : on ne mélange pas la musique et les bruits malgré l'opinion du Roi des Belges Albert 1er (1))
- OC est le numéro d'octave dans l'échelle bien connue des musiciens : l'octave du milieu du piano est 3. Les numéros acceptés par l'ORIC vont de Ø à 6.
- NO est le numéro de note de 1 à 12 avec la correspondance :

1	DO	5	MI	9	SOL#
2	DO#	6	FA	1Ø	LA
3	RE	7	FA#	11	LA#
4	RE#	8	SOL	12	SI

- V est le volume comme pour SOUND, en particulier la valeur Ø pour exécution différée.

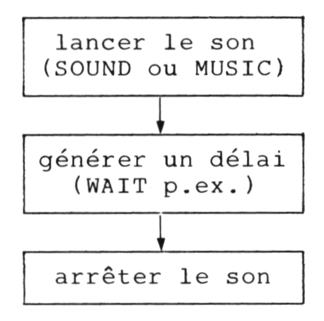
Exemple : MUSIC 1,3,1Ø,1Ø donne le la du violon.

Seul le canal 1 (ou 4) peut être activé directement. Les autres doivent être combinés en mode différé par PLAY. Les commandes SOUND ou MUSIC ont pour seul effet de lancer le son. Celui-ci continue ensuite. C'est d'ailleurs ce qui permet de faire des accords en lançant plusieurs sons.

---

(1) Lorsque sa femme a fondé le concours "Reine Elisabeth de violon", il aurait déclaré : "la musique est le moyen le plus coûteux de faire du bruit". Il ne connaissait pas les microordinateurs !

Vous pouvez suivre le schéma très simple :



Pour arrêter le son dans un programme, il suffit de faire un SOUND ou un MUSIC avec V=Ø sur la voix concernée.

Si un programme s'arrête par STOP, END ou 'Ctrl' C, alors qu'un son est lancé, le son continue. Pour l'arrêter en mode direct, il suffit de taper sur une touche quelconque, du moins quand on est dans le mode où les touches produisent un bip. On passe du mode bip au mode touches silencieuses et vice-versa en faisant 'Ctrl' F.

Pour combiner des sons, la commande PLAY est plus facile et surtout plus riche de possibilités. Elle est de la forme :

PLAY S,B,E,D
où • S (de Ø à 7) indique la combinaison de voix de son pur activées :

$S = \sum_{k \text{ actives}} 2^k$ c'est-à-dire :

S	*voix actives*	*S*	*voix actives*
Ø	aucune	4	3
1	1	5	3 et 1
2	2	6	3 et 2
3	1 et 2	7	les trois

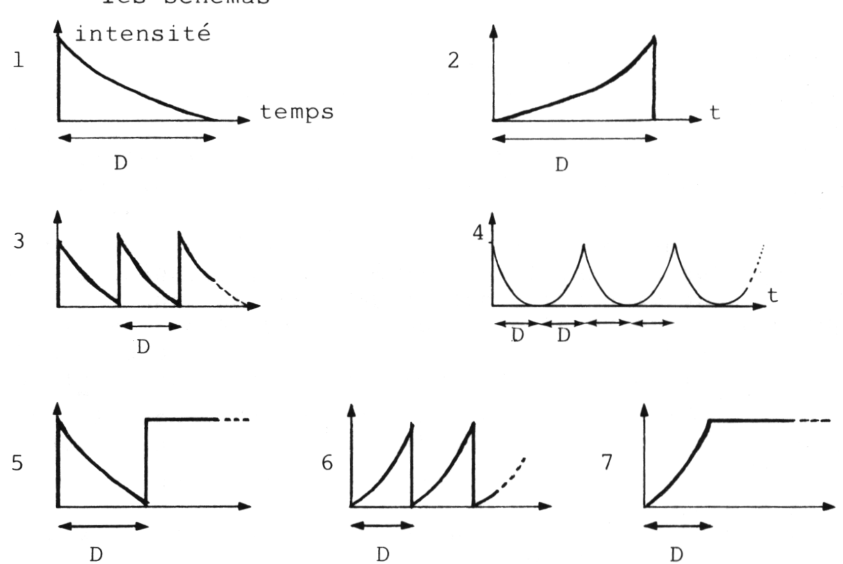
Le son des voix dont on demande l'activation doit préalablement avoir été défini par un SOUND ... Ø, ou un MUSIC ..., Ø.

- B (de Ø à 7) indique la combinaison de voix bruitées activées :

$$B = \sum_{k \text{ actives}} 2^{k-3}$$

Le fonctionnement est le même que pour S et la définition préalable du bruit est tout aussi obligatoire.

- E (de 1 à 7) définit l'enveloppe du son suivant les schémas



Les modes 1 et 2 sont finis dans le temps. Les autres sont de durée illimitée (= jusqu'à ce qu'on arrête le son). Les modes 3, 4 et 6 sont périodiques. 5 est le plus curieux : un coup, puis le son continu. Le mode 1 avec une durée D assez longue est valable pour imiter le piano. Avec une durée très courte il imite un des modes du clavecin. 5 ou 7 avec D très petit donnent un son continu valable pour l'orgue ou la flûte.

- D (de Ø à 32767) définit la durée telle qu'elle est figurée sur les schémas ci-dessus. Donc pour les modes périodiques c'est la période (la demi-période pour 4). Pour les autres, c'est le temps de croissance ou décroissance. (Bien sûr, cette période de l'intensité n'a rien à voir avec la période qui définit la hauteur du son).

D est exprimée en demi-millisecondes. Si D=1ØØØ on a un temps de ½ seconde. Si D<5Ø pour les modes 1 ou 2, on n'entend rien.

Pour mieux nous habituer aux différents effets possibles, nous implantons le programme suivant :

```
  ESSAIS D'ENVELOPPES

1Ø INPUT"E,D";E,D
2Ø MUSIC 1,3,1Ø,Ø
3Ø PLAY 1,Ø,E,D
4Ø GET A$:GOTO 1Ø
```

Exercice 8.1

Quelle note obtient-on ? Que fait l'instruction 4Ø ?

***Question** : Dans le couple SOUND...Ø ou MUSIC...Ø, PLAY il n'y a aucun réglage de l'intensité sonore ?*

- En effet. Le palier ou le maximum de l'enveloppe correspond au réglage 15. Pour un mode 1 ou 2 on peut jouer entre la durée D et la durée réelle du son.

LES BRUITS

Là encore, l'expérimentation sera notre meilleur outil. Pour nous y aider, nous introduisons le programme :

```
  ESSAIS DE BRUITS

1Ø INPUT"P,E,D";P,E,D
2Ø SOUND 4,P,Ø
3Ø PLAY Ø,1,E,D
4Ø GET A$:GOTO 1Ø
```

Pour P, les seules valeurs utiles vont de Ø à 63 avec :

Ø,1	sifflements
2,3,4	réacteurs, fuites de gaz
5 - 8	tempête, lance flamme
9 - 16	vent
17 - 3Ø	machines, explosion
31 - 63	avions, grondements.

Il faut aussi jouer sur E et D : ainsi la combinaison
P=31, E=1, D=5ØØØ figure assez bien une explosion
P=31, E=1, D=1ØØØ figure assez bien un coup de feu

63,4,8Ø   figure assez bien un petit avion.
31,4,1ØØØØ figure les vagues de l'océan...

A vous d'expérimenter !

Il faut aussi essayer du côté des sons purs, des mélanges de voix. Par exemple, deux voix à l'octave l'une de l'autre donnent un son très suave :

```
essayez 1Ø MUSIC 1,3,1Ø,Ø
        2Ø MUSIC 2,4,1Ø,Ø
        3Ø PLAY  3,Ø,7,1
```

Exercice 8.2

Produire l'accord DO MI SOL.

Exercice 8.3

Dans le jeu de Quiz-Géo, quand le joueur a gagné, faites retentir les premières notes de la Marseillaise. (ré ré, ré sol, sol la, la ré, si sol).

Exercice 8.4 *(jeu de SIMON - marque déposée)*

On établit une correspondance entre les touches 1 à 7 et les notes de la gamme. Le programme choisit au hasard une séquence de 4 notes : il la fait retentir tout en affichant sur tout l'écran les couleurs correspondantes. Le joueur doit alors taper la séquence correspondante. Pendant qu'il appuie sur une touche le programme fait retentir la note et affiche la couleur. On autorise deux tentatives pour trouver une séquence.

RECAPITULATION

Nous avons maintenant à peu près exploré toutes les ressources spectaculaires de l'ORIC.

Nous voici arrivés au terme de ce livre, au terme de notre découverte de l'ORIC.

Il faut convenir que c'est là une merveilleuse machine, surtout en regard de son prix.

Ses possibilités sont immenses ; nous n'en avons découvert qu'une petite partie, tant en calcul, en gestion, en machine de loisirs ou d'éducation : notre programme de quiz géographique vous permet de réviser votre géographie en vous amusant... Les possibilités de traitement graphique sont spécialement utiles dans ce contexte, et spécialement perfectionnées sur l'ORIC.

Nous laissons maintenant le champ libre à votre imagination pour en tirer le meilleur parti.

ANNEXE I

Fonctions et mots-clés du Basic

FONCTIONS ARITHMETIQUES

ABS Valeur absolue de l'argument entre parenthèses.

ATN Arc tangente - le résultat est en radians, compris entre -PI/2 et PI/2.

COS Cosinus - l'argument doit être en radians.
Exemple : cos(x en degrés) = COS (PI*X/18Ø).

DEEK fournit le contenu (compris entre Ø et 65535) du double octet d'adresse indiquée, considéré comme un entier 16 bits, dans l'ordre partie basse, partie haute.
Pour écrire un double octet en mémoire, voir DOKE page 145.

EXP Exponentielle e^x. L'argument doit être <88, sinon il se produit un dépassement de capacité.

FALSE Valeur Ø ("fausse" dans les expressions logiques).

FRE Quelle que soit la valeur de l'argument, fournit le nombre d'octets restés libres en mémoire.
Exemple : juste après la mise sous tension, ?FRE (Ø) fournit 39421 (46588 après GRAB). ?FRE("") force un ménage dans la mémoire.

HEX$ convertit son argument (compris entre Ø et 65535) en hexadécimal.

INT Partie entière, ou plutôt plus grand entier inférieur ou égal à l'argument : INT(Ø.5) vaut Ø ; (INT(5) vaut 5, INT(-Ø.5) vaut -1, INT(-3) vaut -3.

KEY$ (pas d'argument). Fournit le caractère actuellement tapé au clavier.

LN Logarithme naturel (népérien, ou en base e).
Pour obtenir le logarithme de X en base Y, utiliser LN(X)/LN(Y).

LOG Logarithme en base 1Ø.

PEEK Fournit le contenu (compris entre Ø et 255) de la case mémoire dont l'adresse est égale à son argument (qui doit être entier et compris entre Ø et 65535).
Pour écrire en mémoire, voir POKE, page 148.

PI Valeur de π.

POINT Renvoie Ø si le point élémentaire indiqué est éteint, -1 s'il est allumé.

POS POS(Ø) fournit la prochaine position d'impression libre sur la ligne d'écran (position du curseur).

RND Fournit un nombre pseudo-aléatoire compris entre Ø et 1. Voir l'explication détaillée au chapitre V.

SCRN Renvoie le code-caractère présent sur l'écran aux coordonnées X et Y fournies comme arguments.

SGN Fonction "signe" : 1 si X>Ø, -1 si X<Ø et Ø si X=Ø.

SIN Sinus - l'argument est supposé en radians.

SPC Ne peut s'employer que dans une instruction PRINT. PRINT SPC(I) imprime I espaces. I doit être entier compris entre Ø et 255.

SQR Racine carrée. L'argument doit être supérieur ou égal à Ø.

TAB Ne peut s'employer que dans une instruction PRINT. PRINT TAB(I); fait aller à la position d'impression n° I(Ø est la position la plus à gauche d'une ligne, 39 la plus à droite). I doit être compris entre Ø et 255. Si I<position où l'on est déjà, il n'y a pas d'action, c'est-à-dire que la prochaine impression se fera là où on était positionné. ***Remarque :*** cette fonction ne marche pas sur les versions actuelles de l'ORIC-1, mais marche sur l'ATMOS.

TAN Tangente - l'argument est supposé en radians.

TRUE Valeur -1 ("Vrai" dans les expressions logiques).

USR Appel d'un programme utilisateur en langage-machine. (avec transmission d'un argument, à la différence de SYS).

FONCTIONS CHAINES

Les fonctions portant sur les chaînes de caractères sont décrites dans le chapitre V.

Nous donnons ci-après leur liste pour mémoire.

LEN(X$) Longueur
LEFT$(X$,N) Extraction à gauche
RIGHT$(X$,N) Extraction à droite
MID$(X$,K) ou **MID$(X$,K,N)** extraction au milieu
ASC(X$) Conversion de caractère en ASCII

CHR$(K) Conversion d'ASCII en caractère
STR$(A) Représentation d'un nombre
VAL(X$) Valeur représentée par une chaîne.

MOTS-CLES RESERVES EN BASIC

ABS	EXPLODE	MUSIC	RND
AND	FILL	NEW	RUN
ASC	FN	NEXT	SCRN
ATN	FOR	NOT	SGN
CALL	FRE	ON	SHOOT
CHAR	GET	OR	SIN
CHR$	GOSUB	PAPER	SOUND
CIRCLE	GOTO	PATTERN	SPC
CLEAR	GRAB	PEEK	SQR
CLOAD	HEX$	PING	STEP
CLS	HIMEM	PLAY	STOP
CONT	HIRES	PLOT	STR$
COS	IF	POINT	TAB
CURMOV	INK	POKE	TAN
CURSET	INPUT	POP	TEXT
CSAVE	INT	POS	THEN
DATA	LEFT$	PRINT	TO
DEEK	LEN	PULL	TROFF
DEF	LET	READ	TRON
DIM	LIST	RELEASE	UNTIL
DOKE	LLIST	REM	USR
DRAW	LN	REPEAT	VAL
ELSE	LOG	RESTORE	WAIT
END	LORES	RETURN	ZAP
EXP	MID$	RIGHT$	

VARIABLES RESERVEES : FALSE PI TRUE

ANNEXE II

Répertoire des Instructions et des Opérateurs Basic

Comme le Guide Michelin, nous attribuons des étoiles aux instructions et aux commandes : *** instruction fondamentale, ** instruction importante, * instruction intéressante. Pas d'* : instruction qu'il ne faut pas hésiter à utiliser si le besoin s'en fait sentir ; certaines de ces instructions ne sont pas traitées dans ce livre.

D'autre part, P veut dire : plutôt instruction en mode programmé, P̸ veut dire : interdit en mode programmé, C : plutôt commande en mode direct, C̸ : interdit en mode direct, "rien" veut dire : tantôt l'un, tantôt l'autre.

La définition est suivie d'exemples non commentés montrant les différentes formes que peut prendre l'instruction.

Catégorie	*Mot-clé*	Définition-*exemple*	*Page*
	CALL	Fait démarrer l'exécution d'un programme en langage machine à l'adresse indiquée *25 CALL 62512*	
*	**CHAR**	Dessine un caractère à l'emplacement du curseur haute résolution CHAR code,jeu, allumé *1ØØ CHAR 65,Ø,1*	102
	CIRCLE	Dessine un cercle centré à l'emplacement du curseur : CIRCLE rayon, allumé. *1ØØ CIRCLE 5Ø,1*	102
C	**CLEAR**	Remise à zéro de toutes les variables.	33
**	**CLS**	Vide l'écran	33
C**	**CLOAD**	Chargement d'un programme sur cassette *CLOAD"" CLOAD"NOM",S*	45
CP̸	**CONT**	Continuer dans le programme après interruption.	24

Catégorie	*Mot-clé*	Définition- *exemple*	*Page*
*	**CURMOV**	Déplace le curseur haute résolution : CURMOV DX,DY, allumé. *1ØØ CURMOV 1Ø,1Ø,Ø*	102
**	**CURSET**	Positionne le curseur haute résolution : CURSET X,Y allumé. *1ØØ CURSET Ø,Ø,1*	102
C**	**CSAVE**	Sauvegarde d'un programme sur cassette. *CSAVE"NOM" CSAVE"NOM",S CSAVE"NOM",AUTO*	45
P*	**DATA**	Définit une liste de constantes qui seront "lues" par une instruction READ. *1Ø DATA ABC,DEF,5,3,25*	72
P*	**DEF FN**	Définition d'une fonction utilisateur. *1Ø DEF FNF(X)=A*X+B*	90
	DEF USR	Définit l'adresse de la fonction USR. *DEF USRY=4ØØ*	
P*	**DIM**	Dimensionnement d'un tableau (fixe les valeurs maximum des indices). *1Ø DIM A(15),B(3), C$(2,5),D(7,6,2) - 2Ø DIM W(N)*	75
*	**DOKE**	DOKE a,b écrit la donnée b dans le double octet d'adresses a (poids faibles) et a+1 (poids forts) *DOKE 63Ø,65535 DOKE A,X-4*	65
**	**DRAW**	Dessine un vecteur à partir de la position du curseur haute résolution, de composantes DX,DY : DRAW DX,DY, allumé.	102
**	**ELSE**	Introduit l'instruction à effectuer lorsque la condition d'un IF n'est pas satisfaite. *IF X<Ø THEN A=B ELSE A=X*	52
PȻ*	**END**	Termine un programme.	35
	EXPLODE	Produit un bruit d'explosion.	133

Catégorie	*Mot-clé*	Définition-*exemple*	*Page*
	FILL	FILL NLE,NC,Z remplit NLE lignes élémentaires sur NC cases avec le code Z à partir du curseur haute résolution. *FILL 2ØØ,4Ø,63*	130
**	**FOR**	Introduit une boucle : toutes les instructions comprises entre FOR I = A TO B STEP C et le NEXT correspondant seront répétées pour toutes les valeurs de I allant de A à B, C par C: *1Ø FOR I=1 TO 1ØØØ* *5Ø FOR I=Ø TO 2ØØ STEP 5* *6Ø FOR I=N TO 3*N+4 STEP 5* *7Ø FOR I=5Ø TO 1 STEP -1* *8Ø FOR X=1.5 TO 2*PI STEP .1*	58
P₡*	**GET**	Attend un caractère au clavier *1Ø GET A$*	106
P**	**GOSUB**	Appel d'un sous-programme *1Ø GOSUB 1ØØØ*	120
P**	**GOTO**	Saut à une autre instruction *1Ø GOTO 5Ø*	24
	GRAB	Récupère pour Basic la zone mémoire d'écran haute résolution.	108
	HIMEM	Abaisse le sommet de mémoire connu de Basic : permet de protéger une zone mémoire pour un sous-programme en langage machine. *HIMEM #8ØØØ*	
**	**HIRES**	Passe en mode haute résolution	92
P**	**IF**	Saut conditionnel, de la forme : IF condition THEN instruction. Si la condition n'est pas satisfaite, on passe à la ligne suivante ; si la condition est satisfaite, on effectue l'instruction qui suit THEN. IF c THEN GOTO s'élide en IF c THEN x ou en IF c GOTO x *1Ø IF A>B THEN Y=K*	50

Catégorie	_Mot-clé_	Définition-_exemple_	_Page_
		2Ø IF A=3 GOTO 1ØØØ _3Ø IF A$<>"" THEN 5Ø_ Autre forme : IF condition THEN i1 ELSE i2 si la condition est satisfaite on effectue i1, sinon i2. _IF A=3 THEN Y=12 ELSE Y=1Ø_	
*	**INK**	Définit la couleur des tracés ("encre") _INK 5 INK 1+INT(7*RND(1))_	100
P¢***	**INPUT**	Acquisition de données au clavier _1Ø INPUT A_ _2Ø INPUT A,B,C$,D_ _3Ø INPUT"ENTREZ UN NOMBRE";N_	22,26
C***	**LIST**	Liste du programme _LIST LIST 1Ø LIST-1ØØ_ _LIST 1Ø LIST 1Ø-1ØØ_	31
	LLIST	Listing sur l'imprimante.	157
	LORES	Passage en mode basse résolution. _LORES Ø LORES 1_	92
	LPRINT	Ecriture sur l'imprimante.	157
**	**MUSIC**	MUSIC NV,OC,NO,V prépare ou fait retentir une note sur la voix NV, l'octave OC, de hauteur définie par NO et de volume V. _MUSIC 1,4,1Ø,1Ø_	134
¢F*	**NEW**	Vide la mémoire-programme.	33
**	**NEXT**	Fait passer à l'itération suivante dans un FOR _NEXT I NEXT J,I NEXT_	58
P	**ON**	_ON I GOTO 1Ø,2Ø,3Ø_ Si I vaut 1, on va en 1Ø, s'il vaut 2, on va en 2Ø, en 3Ø s'il vaut 3. _ON I GOSUB 1ØØØ,15ØØ,2ØØØ,5ØØØ_ Si I vaut 1, on appelle le sous-programme en 1ØØØ, 2 en 15ØØ, 3 en 2ØØØ, 5 en 5ØØØ	

Catégorie	*Mot-clé*	Définition-*exemple*	*Page*
*	**PAPER**	Définit la couleur du fond de l'écran ("papier"). *PAPER 7 PAPER NP(I)*	100
	PATTERN	Définit le motif des pointillés pour les tracés en haute résolution. *PATTERN M*	102
	PING	Fait retentir un "ding" aigu	133
*	**PLAY**	PLAY S,B,E,D exécute les sons préparés préalablement par SOUND ou MUSIC avec la combinaison S (de Ø à 7) de voix musicales, B (de Ø à 7) de voix bruitées, l'enveloppe E (de 1 à 7) et caractérisée par la durée D (de 1 à à 32767 en demi-millisecondes). *PLAY 7,Ø,7,1ØØØ*	135
**	**PLOT**	PLOT X,Y,A$ écrit la chaîne A$ à partir de la position X,Y sur l'écran PLOT X,Y,A écrit le caractère ou l'attribut de code A *PLOT 1Ø,1Ø,12 PLOT 2Ø,15, PLOT 2Ø, 15, "BONJOUR"*	94
*	**POKE**	POKE a,b écrit la donnée b à l'adresse a. *POKE 36879,27 POKE K+1,Z-4* *Pour lire en mémoire, voir PEEK.*	65
	POP	Enlève une adresse de retour de la pile.	
*	**PRINT**	Imprime un résultat sur l'écran. *PRINT A 1Ø PRINT A;B,J* *2Ø PRINT 2*A+3,B$* *3Ø PRINT"LE RESULTAT EST";B;*	15,27
	PULL	Enlève une adresse de boucle de la pile.	
P*	**READ**	"Lecture" de données dans une instruction DATA associée. *1Ø READ A 2Ø READ A,B$,C*	72

Catégorie	*Mot-clé*	Définition-*exemple*	*Page*
**	**RECALL**	Lecture de données sur cassette *1ØØ RECALL A,"ZOZO",S* *11Ø RECALL X$,"FICH"*	157
	RELEASE	Fait rendre par Basic l'espace mémoire de l'écran haute résolution. Nécessaire pour faire HIRES après un GRAB.	108
P	**REM**	Introduit un commentaire.	77
**	**REPEAT**	REPEAT : instructions :...: UNTIL condition Fait répéter les instructions entre le REPEAT et UNTIL jusqu'à ce que la condition soit réalisée. *REPEAT:GET A$:UNTIL A$=CHR$(13)*	63
P	**RESTORE**	Revient au début des DATA.	72
P**	**RETURN**	Retour de sous-programme. *1ØØ RETURN*	120
C***	**RUN**	Déclenche l'exécution d'un programme. *RUN RUN 5ØØ*	16,34
	SHOOT	Produit un bruit de coup de feu	133
*	**SOUND**	SOUND NV,P,V fait retentir un son par le canal NV (de 1 à 3, ou, avec bruit, de 4 à 6), de fréquence 62ØØØ/P, de volume V (Ø à 15). *SOUND 1,14Ø,1Ø*	133
**	**STEP**	Introduit le pas d'incrémentation dans un FOR.	60
P¢	**STOP**	Arrête l'exécution d'un programme. *1Ø STOP*	68
**	**STORE**	Stockage d'un tableau de données sur cassette *1ØØ STORE A,"ZOZO",S*	157
**	**TEXT**	Passe en mode TEXT	92
P**	**THEN**	Introduit l'instruction à effectuer quand un IF est satisfait.	50
**	**TO**	Introduit la valeur limite d'un FOR.	58

Catégorie	*Mot-clé*	Définition-*exemple*	*Page*
*	**TROFF**	Arrête la trace.	69
*	**TRON**	Met en oeuvre la trace (liste des n° d'instructions par où l'on passe).	69
**	**UNTIL**	Introduit la condition d'arrêt d'une boucle REPEAT.	63
*	**WAIT**	WAIT N fait attendre N centièmes de seconde.	66
	ZAP	Produit un bruit de tir d'un canon laser.	133
***	=	Affectation de valeur à une variable. *A=36*	17
	#	Dénote une constante hexadécimale *?DEEK (#CØØØ).*	

OPERATEURS

Arithmétiques
+ addition de nombres ou concaténation de chaînes de caractères
- prendre l'opposé ou soustraction
* multiplication
/ division

logiques effectués bit à bit		
NOT	non logique, agit sur 1 seul opérande	
AND	et logique	2 opérandes
OR	ou logique	2 opérandes

de relation	
= égal	<> différent
< inférieur	> supérieur
<= inf. ou =	>= sup. ou =
=< inf. ou =	=> sup. ou =

bit 1	bit 2	AND	OR
Ø	Ø	Ø	Ø
Ø	1	Ø	1
1	Ø	Ø	1
1	1	1	1

bit	NOT
Ø	1
1	Ø

ANNEXE III

MESSAGES D'ERREUR

Le fait de disposer d'un interpréteur qui prend les instructions Basic une par une pour les exécuter permet, lorsque se produit une erreur, d'identifier, avec précision, l'instruction où s'est produit l'incident.

Par suite, les messages d'erreur seront de la forme :

? message ERROR in numéro
précisant le numéro de la ligne où l'erreur a été rencontrée. Le seul cas où le numéro n'est pas précisé est celui d'une commande directe.
Les problèmes les plus courants proviennent d'un mot-clé mal orthographié, d'une virgule manquante ou d'une variable d'un certain type employée à tort, etc...
Il faut noter qu'une erreur signalée à un numéro de ligne peut n'être que la conséquence d'une autre erreur : par exemple une DIVISION BY ZERO ERROR vient plutôt des instructions précédentes où le diviseur est calculé.

Nous donnons, ci-dessous, la traduction, et pour les plus importants, des explications, des différents messages d'erreur qui peuvent être délivrés par l'ORIC lors de l'exécution d'un programme Basic.

BAD SUBSCRIPT *(mauvais indice)*
Tentative d'appeler un élément de tableau d'indice supérieur à la limite fournie dans le DIM, ou encore avec un nombre d'indices différent de celui spécifié dans le DIM.
Exemple : DIM A(15) avec A(2Ø) ou A(2,2).

BAD UNTIL *(mauvais UNTIL)*
UNTIL rencontré alors qu'il n'y a pas eu de REPEAT.

CAN'T CONTINUE *(on ne peut pas continuer)*
Impossibilité de reprendre l'exécution d'un programme par CONT. C'est le cas s'il y a eu une erreur, ou si, pendant l'interruption, on a modifié ou ajouté une instruction. On peut reprendre par GOTO numéro dans certains cas.

DISP TYPE MISMATCH *(Désaccord avec le mode d'affichage)*
Emploi d'une instruction caractéristique d'un mode d'affichage alors qu'on est dans un autre.
Exemple : DRAW alors qu'on est en mode TEXT.

DIVISION BY ZERO *(Division par zéro)*
Vient le plus souvent d'une variable non initialisée.

FORMULA TOO COMPLEX *(Expression chaîne de caractères trop complexe ou trop de IF...THEN sur une ligne).*

ILLEGAL DIRECT *(illégal en mode direct)*
INPUT, GET et DEF FN sont interdites en mode direct.

ILLEGAL QUANTITY *(valeur erronée)*
Emploi d'une fonction avec un argument hors de l'intervalle permis. Exemples :

- indice négatif ou >32767
- argument de LOG négatif ou nul
- argument de SQR négatif
- Ø puissance nombre négatif
- longueur spécifiée dans MID$, RIGHT$, LEFT$ non comprise entre Ø et 255
- index de ON GOTO hors des limites
- adresses dans PEEK, POKE, DEEK ou DOKE non comprises entre Ø et 65535
- octet spécifié dans POKE, TAB, ou SPC non compris entre Ø et 255.

NEXT WITHOUT FOR *(NEXT sans FOR correspondant)*
Vient le plus souvent de boucles mal imbriquées, d'une confusion sur la variable marquée dans le NEXT ou encore si l'on a supprimé un FOR pour une correction, en oubliant de supprimer le NEXT.

OUT OF DATA *(DATA épuisées)*
On essaie de faire un READ alors que l'on a déjà "lu" toutes les données de toutes les DATA. Il faut, soit comptabiliser les données avec soin, soit utiliser RESTORE.

OUT OF MEMORY *(mémoire épuisée)*
Programme trop long, ou trop de variables, ou trop de boucles et GOSUB imbriqués.

OVERFLOW *(Dépassement de capacité)*
Résultat d'un calcul supérieur à 1.7E38. Dans l'autre sens, un résultat inférieur à 2.9E-39 est indiscernable de Ø, mais il n'y a pas de message d'erreur.

REDIM'D ARRAY *(Tableau redimensionné)*
On est repassé une deuxième fois sur l'instruction DIM portant sur un tableau, ou il y a une deuxième instruction DIM pour un tableau.

REDO FROM START *(Recommencez depuis le début)*
Lors d'une instruction INPUT, on a fourni une quantité alphanumérique, alors que l'on s'attendait à du numérique. Il faut reprendre en redonnant toutes les valeurs attendues par l'instruction INPUT considérée.

RETURN WITHOUT GOSUB *(Retour sans GOSUB)*
On est tombé sur un RETURN alors que l'on ne venait pas d'exécuter un GOSUB. Dû le plus souvent à l'oubli de END en fin de programme principal qui précède un sous-programme.

STRING TOO LONG *(Chaîne de caractères trop longue)*
Tentative de créer (par concaténation) une chaîne de plus de 255 caractères.

SYNTAX *(Erreur de syntaxe)*
Instruction incompréhensible pour Basic. Dû notamment à des parenthèses non appariées, caractères illégaux, mauvaise ponctuation, faute d'orthographe dans un mot-clé.

TYPE MISMATCH *(Désaccord entre numérique et alphanumérique)*
Valeur numérique affectée à une variable chaîne, ou vice-versa, ou bien argument numérique fourni à une fonction qui demande un argument alphanumérique, ou vice-versa.

UNDEF'D STATEMENT *(Instruction non définie)*
GOTO, GOSUB ou THEN renvoyant à un numéro de ligne inexistant. C'est le plus souvent dû à l'oubli de correction d'un GOTO... quand la ligne cible a été supprimée ou changée de place.

UNDEF'D FUNCTION *(Fonction non définie)*
Référence à une fonction pour laquelle on a oublié le DEF FN.

L'erreur ***FILE ERROR LOAD ABORTED*** a été traitée au chapitre III dans la section sur les cassettes.

ANNEXE IV

QUESTIONS ET REPONSES

Lorsque je veux relancer un programme après interruption (en 1ØØ), je peux utiliser CONT,RUN 1ØØ ou GOTO 1ØØ (en mode direct). Quelle est la différence ?

CONT ne peut pas être utilisé si, au cours de l'interruption, vous avez apporté une correction au programme. Si vous utilisez RUN 1ØØ, toutes vos variables seront remises à zéro. Donc, si lors de l'interruption, vous avez corrigé une variable, il faut utiliser GOTO 1ØØ.

Lorsque la variable à lire par INPUT est une chaîne de caractères, doit-on la mettre entre guillemets ?

En principe non. On la met entre guillemets si la chaîne contient des virgules, deux-points, ou des espaces au début ou à la fin.

Que se passe-t-il lorsqu'en INPUT d'un nombre on fournit une chaîne de caractères, ou vice-versa ?

Si à INPUT A$ vous répondez 123, l'ORIC prendra la chaîne de caractères chiffre 1, chiffre 2, chiffre 3 ; pas de problème.

Si à INPUT A vous répondez ABC, il y aura une erreur avec le message REDO FROM START et l'instruction INPUT sera réexécutée ; donc, pour un INPUT à plusieurs variables, vous devez refournir toutes les données.

Y-a-t'il une limite à la longueur des chaînes de caractères ?

Oui : 255 caractères, ce qui est déjà beaucoup. Notez qu'une telle chaîne ne peut pas être donnée d'un seul coup, puisqu'une instruction ne peut dépasser 8Ø caractères ; elle doit être construite par concaténation.

Comment prévoir la taille d'un programme ?

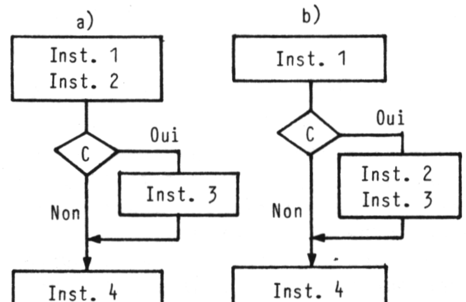
En gros, un programme occupe autant d'octets qu'il renferme de caractères, puisqu'il est stocké tel quel, comme chaîne de caractères, sauf les mots-clés qui sont remplacés par un code en 1 octet.

En ce qui concerne les variables, chaque variable numérique occupe 7 octets, chaque chaîne occupe (7 + longueur) octets. Un tableau occupe x (n+1)+2d, où d est le nombre de dimensions, n est la taille du tableau (y compris l'élément n° Ø) et x = 5 (nombres) ou 3 (chaînes de caractères).

On gagne de la place mémoire en supprimant tous les blancs inutiles, en mettant plusieurs instructions par ligne, en évitant les REM, en utilisant des variables plutôt que des constantes. Utilisez des GOSUB dès qu'il faut faire appel plusieurs fois à une séquence d'instructions identiques.

J'ai des problèmes lorsque j'ai plusieurs instructions sur la même ligne, avec un IF parmi elles.

La forme : 1∅ inst.1 : int.2 : IF c THEN inst.3
2∅ inst.4
ne doit vous causer aucun problème. Elle obéit à l'ordinogramme a) ci-dessous :



La forme
5 inst.1
1∅ IF c THEN inst.2 : inst.3
2∅ inst.4
obéit, <u>sur l'ORIC</u>, à l'ordinogramme b) ci-contre. C'est-à-dire que lorsque la condition n'est pas satisfaite, on passe à la ligne suivante et non à l'instruction qui suit l'instruction introduite par THEN.

Que se passe-t-il si je mets une instruction FOR sans NEXT ?

Les instructions qui suivent le FOR sont exécutées une fois (puisque rien ne dit à l'ORIC qu'il faut recommencer). La variable, citée dans le FOR, prend pour valeur la valeur de départ (placée avant TO).

Comment imprimer des nombres alignés sur leur droite ?

Il faut employer les fonctions chaînes de caractères. Par exemple, on dispose d'une zone de 10 caractères dans laquelle on veut imprimer un nombre N. Quel que soit le nombre de chiffres, on veut que le nombre soit imprimé à droite de la zone. On emploie la séquence suivante :

```
1∅∅ N$=STR$(N) :L=LEN(N$)
11∅ IF L=1∅ GOTO 13∅
12∅ FOR I=1 TO 1∅-L:N$=" "+N$ :NEXT
13∅ PRINT N$
```

On peut remplacer 12∅ par 12∅ PRINT SPC(1∅-L);

Je viens de faire un programme que je voudrais sauvegarder sur cassette, mais j'ai oublié de positionner la cassette.

Il ne faudrait surtout pas faire le SAVE avec la cassette rebobinée : vous effaceriez une partie des programmes déjà enregistrés. Si votre magnéto a un compteur et si vous avez noté les numéros, vous pouvez vous positionner derrière le dernier programme qui avait été enregistré sur la cassette. Sinon, prenez une autre cassette.

Comment savoir ce qu'il y a sur une cassette ?

Il faut tenir un répertoire des cassettes à mesure qu'on les remplit. Sinon, lire chaque programme par CLOAD"" (sans préciser le nom).

J'ai vu, sur certains listings, des noms de variables se terminant par %. Qu'est-ce que c'est ?

Il s'agit d'un type de variable que nous n'avons pas étudié : les variables entières. Leurs noms sont de la forme A% A1% ALBERT%. On peut, dans un programme, avoir les trois variables distinctes A% A et A$. Les variables % peuvent être dimensionnées. Ces variables ne peuvent prendre que des valeurs entières comprises entre -32767 et +32767. Elles sont intéressantes pour économiser de la mémoire, étant donné qu'elles n'occupent que deux octets.

Est-ce que, à part des programmes, on peut écrire des données sur cassette ?

Oui, mais le Basic de l'ORIC n'offre aucune facilité pour cela. Il faut recourir à des techniques délicates qui seront décrites dans le prochain livre de cette série, sur les périphériques et fichiers. Il y aura aussi des articles sur cette question dans "La Commode".

Le programme suivant est le squelette d'une gestion de fichiers sur cassette. Lorsqu'on fait RUN (le magnéto étant en écriture), on acquiert au clavier des chaînes de caractères qu'on transfère sur cassette. On termine en fournissant $$$ comme chaîne. Lorsqu'on fait RUN 40, les données sont relues et imprimées sur l'écran.

```
10 INPUT A$
20 GOSUB 1000:GOSUB 1050
30 IF A$<>"$$$" GOTO 10
35 END
40 GOSUB 1030:GOSUB 1060
50 A$="":A=1024
60 B=PEEK(A):IF B=0 GOTO 80
70 A$=A$+CHR$(B):A=A+1:GOTO 60
```

```
80 PRINT A$:IF A$<>"$$$" GOTO 40
90 END
1000 LL=LEN(A$)+1:FOR I=1 TO LL-1
1010 POKE 1023+I,ASC(MID$(A$,I,1)):NEXT:POKE 1023+LL,0
1020 DOKE #5F,#400:POKE #61,LL:POKE #62,4
1030 DOKE #63,#100:POKE #67,0
1040 DOKE #35,65:RETURN
1050 CALL #E6CA:CALL #E57B:CALL #E804:RETURN
1060 CALL #E6CA:CALL #E4A8:CALL #E804:RETURN
```

Pour l'utilisation de ce programme, il est préférable de disposer de la télécommande du magnéto. Cette procédure est inutile avec l'ORIC ATMOS qui dispose d'instructions spéciales pour cela :
STORE V,"NOM"[,S] range le tableau V dans le fichier NOM sur cassette
RECALL V,"NOM"[,S] récupère dans le tableau V les données du fichier NOM.

|J'ai une imprimante. Comment l'utiliser ?

1°) Pour obtenir un listing sur l'imprimante, faire LLIST.

2°) Pour écrire sur l'imprimante, faire LPRINT en lieu et place de PRINT.

Selon l'imprimante, des LPRINT CHR$(x) peuvent avec certaines valeurs de x envoyer des caractères de contrôle déclenchant un comportement spécial de l'imprimante, comme saut de page, impression en caractères larges etc... Reportez-vous à la notice de l'imprimante. L'imprimante vous fait des problèmes du style caractères répétés ou manquants, voyez dans "La Commode" n° 9, page 60, une explication possible et le remède correspondant qui est d'interdire les interruptions le temps de l'impression : CALL#ED01:LPRINT.....:CALL#ECC7.

ANNEXE V

SOLUTIONS DES EXERCICES

Les exercices non rappelés ici ont, en principe, leur solution dans le texte.

Exercice 2.1

```
10 INPUT"RAYON",HAUTEUR";R,H
20 V=PI*R↑2*H
30 PRINT"VOLUME=",V
40 GOTO10
100 REM 20 ET 30 POURRAIENT ETRE REMPLACES
110 REM PAR 25 PRINT"VOLUME=",PI*R↑2*H
```

Exercice 2.2

```
10 INPUT"CAPITAL,TAUX EN %,NOMBRE D'ANNEES";C,T,N
20 I=C*((1+T/100)↑N-1)
30 PRINT"INTERET=";I
40 GOTO10
```

Exercice 3.2

Appuyez sur 'Del' deux fois, puis JOUR.

Exercice 3.3

Taper : BONJOUR 'Esc'L 'Esc'G 'Esc'P MADAME 'Esc'W 'Esc'D 'Esc'H BONJOUR 'Esc'L 'Esc'C 'Esc'T MONSIEUR

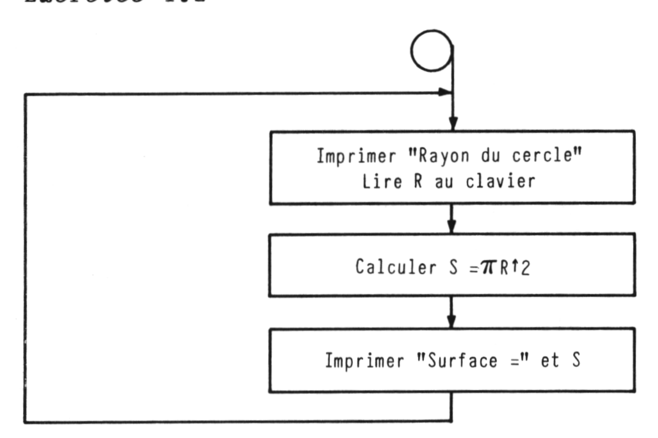
Exercice 3.5

Pour que l'instruction 30 passe de 30 PRINT"SURFACE =", S à 30 PRINT"VOLUME =",V; amener le curseur sur le S de surface par 'Ctrl' A et taper VOLUME=",V 'Return'.

Exercice 4.1

```
            15 IF S<=Ø GOTO 1Ø
Variante :  15 IF S<=Ø GOTO 5Ø
            5Ø PRINT"IL FAUT UNE SURFACE POSITIVE"
            6Ø GOTO 1Ø
```

Exercice 4.2



Imprimer "Surface du cercle"
Lire S au clavier

S > 0

Non

Imprimer "il faut une surface positive"

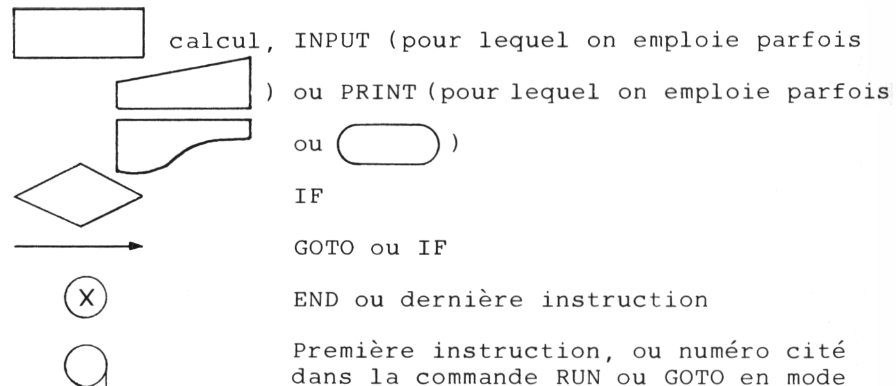
Oui

Calculer R = $\sqrt{\frac{S}{\pi}}$

Imprimer "Rayon", R



Exercice 4.3



Exercice 4.4

```
PRINT INT(X*1Ø↑D)/1Ø↑D
```

Exercice 4.5

Deux solutions :
1. faire l'impression en 65 et mettre 6Ø N=N+1
2. initialiser N à 1, ce qui doit se faire explicitement en ajoutant 1Ø N=1.

Exercice 4.7

Il manque l'impression d'une première ligne de titre !

```
  5 PRINT"NB","CARRE","RACINE"
  6 PRINT
```

(Le 6 fait sauter une ligne).

Exercice 4.8

Seule instruction changée :

```
  1Ø FOR N=2 TO 1Ø STEP 2
```

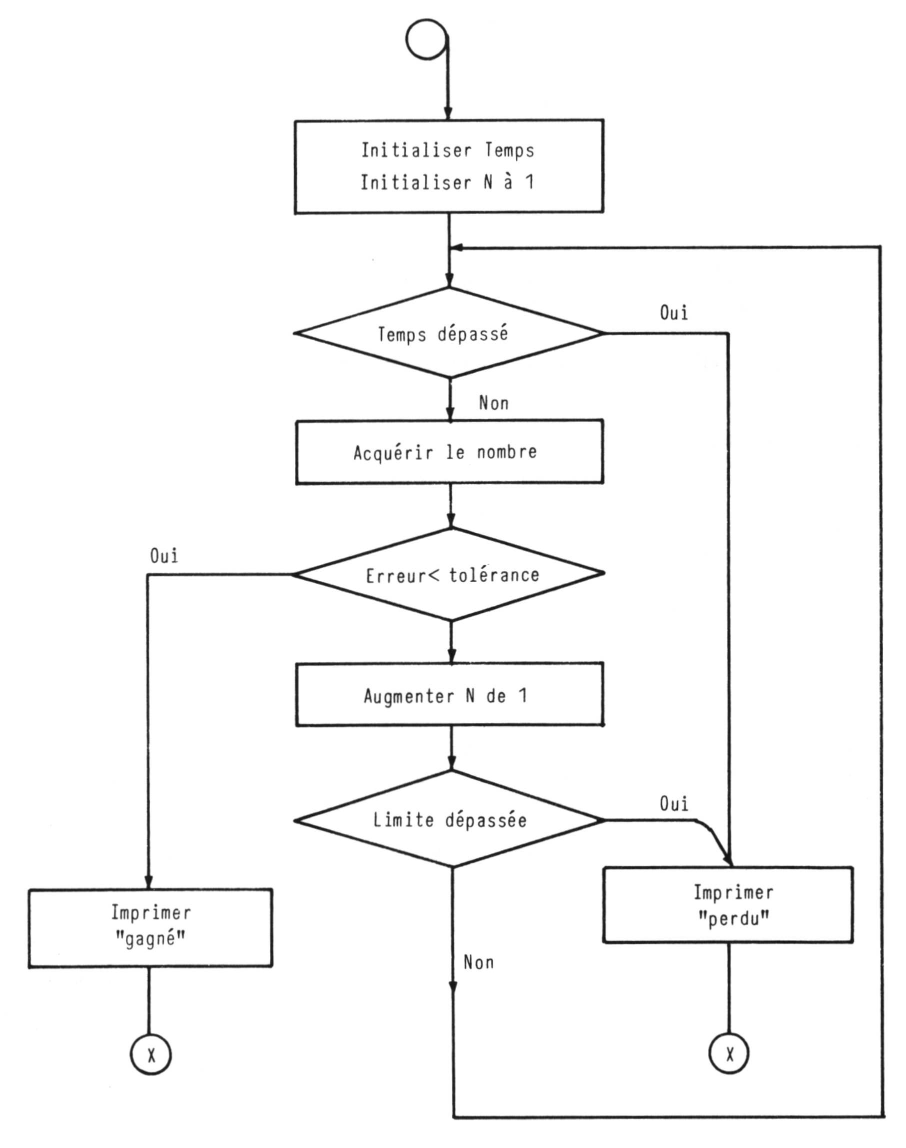
Exercice 4.9

```
  1Ø : 1  3  5  7              valeur finale 9
  5Ø : 1Ø 9  8  7  6  5  4     valeur finale 3
```

Exercice 4.11

2 minutes : on accorde 72ØØ soixantièmes de seconde.

Exercice 4.12



Exercice 4.13

Problème déjà résolu dans l'instruction 4Ø :
6Ø PRINT"GAGNE EN";N;"COUPS ET";INT((65535-DEEK(63Ø)).6)
/1ØØ "SECONDES"

Exercice 4.14

```
15 IF 65535-DEEK(630)>7200 GOTO 55
50 PRINT"NB. D'ESSAIS PERMIS DEPASSE"
53 END
55 PRINT"TEMPS DEPASSE"
57 END
```

Exercice 5.1

```
25 IF C<>99999 GOTO 30
27 RESTORE:GOTO 20
100 GOTO 10
210 DATA 210,78.31,901.5,31,4.18,2.7,99999
```

Exercice 5.2

A la suite du calcul de la moyenne, on aurait :

5-2 A

```
30 V=0
100 FOR I=1 TO N
110 V=V+(A(I)-M)↑2
120 NEXT I
130 V=V/(N-1)
```

On peut proposer une solution plus élégante, qui calcule moyenne et variance dans la même boucle, et qui repose sur la remarque mathématique suivante :

$$(A_i - M)^2 = \sum_i (A_i^2 - 2MA_i + M^2)$$

$$= \sum_i A_i^2 - 2M\sum_i + N*M^2$$

$$= \sum_i A_i^2 - N*M^2$$

D'où le programme :
5-2 B

```
10 DIM A(N)
20 REM LECTURE DES A
30 S=0:S2=0
40 FOR I=1 TO N
50 S=S+A(I):S2=S2+A(I)↑2
60 NEXT I
70 M=S/N:V=(S2-N*M↑2)/(N-1)
80 PRINT"MOYENNE ";M,"VARIANCE ";V
```

Exercice 5.3

```
10 DIM U(N),V(N)
20 REM LECTURE
30 UV=0
40 FOR I=1 TO N
50 UV=UV+U(I)*V(I)
60 NEXT I
```

Exercice 5.5

```
10 REM ON NE S'OCCUPE PAS DU TOUT
15 REM DES ENTREES/SORTIES
20 DIM A(N,N),B(N,N),C(N,N)
30 FOR I=1 TO N
40 FOR J=1 TO N
50 C(I,J)=0
60 FOR K=1 TO N
70 C(I,J)=C(I,J)+A(I,K)*B(K,J)
80 NEXT K:NEXT J:NEXT I
```

Exercice 5.6

```
1Ø X$=LEFT$(X$,4)+"A"+MID$(X$,6)
```

Exercice 5.7

```
100 N=LEN(A$):P=LEN(B$)
110 FOR K=1 TO N-P+1
120 IF MID$(A$,K,P)=B$ GOTO 150
130 NEXT K
140 K=0
150 PRINT K
160 END
```

Lorsque vous aurez vu les sous-programmes, vous comprendrez qu'il est très judicieux de remplacer 160 par 160 RETURN.

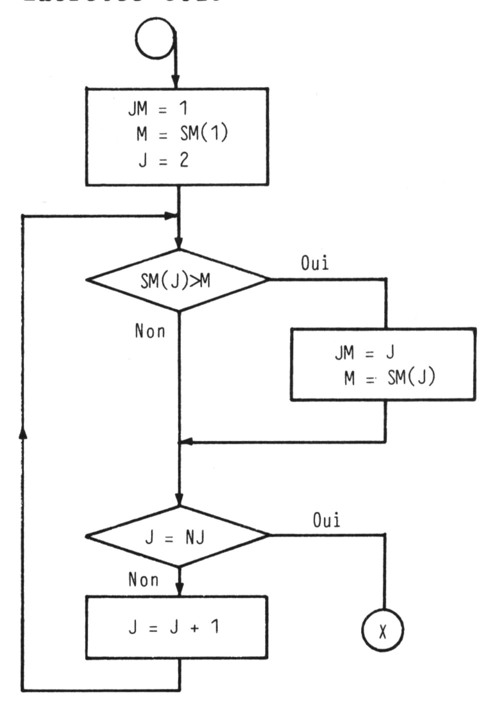
Exercice 5.8

NC = LEN(STR$(A))-1 dans les deux cas : STR$ est corrigée pour cela dans l'ORIC.

Exercice 5.9

```
100 B$=STR$(B):N=LEN(B$)
110 PRINT LEFT$(B$,N-3)
```

Exercice 5.10



On suppose au départ que le maximum est au rang 1.

Ensuite, on parcourt le tableau SM à partir du rang 2. Si on trouve un élément supérieur au maximum supposé, c'est cet élément que l'on prend comme nouveau maximum supposé

On peut alors ajouter au programme B-10 la séquence suivante :

```
210 PRINT:PRINT:DIM SM(NJ)
220 FOR J=1 TO NJ:SM(J)=0
230 FOR P=1 TO NP:SM(J)=SM(J)+SC(J,P)
240 NEXT P:SM(J)=SM(J)/NP:NEXT J
250 JM=1:M=SM(1)
260 FOR J=2 TO NJ
270 IF SM(J)<=M THEN 290
280 JM=J:M=SM(J)
290 NEXT J
300 PRINT"LE GAGNANT EST  ";NOM$(JM)
310 PRINT"AVEC UN SCORE MOYEN DE";M
```

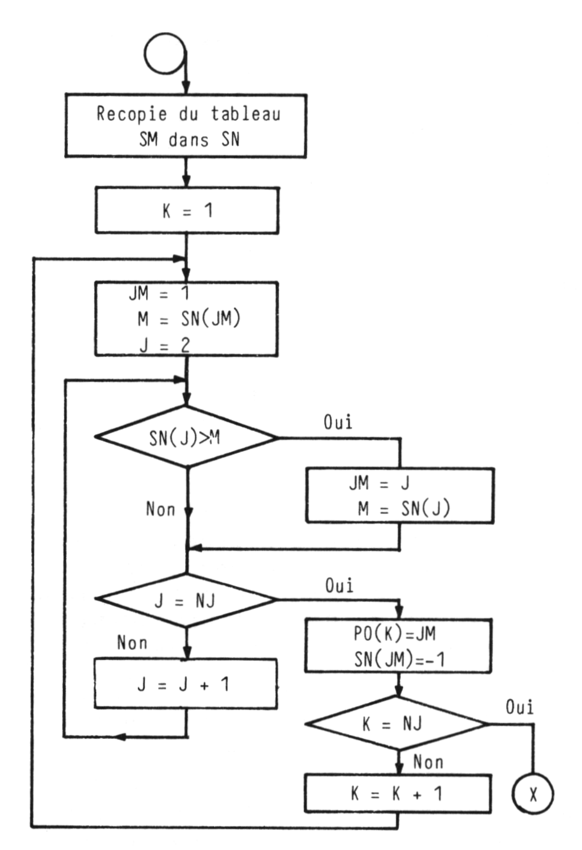
Exercice 5.11

Les problèmes de classement ou de tri sont parmi les plus souvent rencontrés. On distingue les problèmes de tri simple où, partant d'un tableau d'éléments, on cherche à obtenir, à la fin du programme, le même tableau, mais ordonné, les éléments étant maintenant croissants ou décroissants ; et les problèmes de classement où on laisse en place les éléments du tableau de départ, mais on forme un tableau auxiliaire, dit tableau de pointeurs PO(I), tel que PO(I) soit l'indice dans le tableau de départ de l'élément qui mérite d'être classé I ième. Le nom du joueur classé I ième est ainsi NOM$ (PO(I)).

Dès que les éléments sont encombrants, il est plus intéressant d'effectuer un classement, plutôt qu'un tri simple : les termes à déplacer sont plus petits.

Une méthode de classement simple découle de l'exercice précédent. Supposons qu'on veuille un classement par ordre de score moyen décroissant. PO(1) n'est rien d'autre que le JM obtenu précédemment. PO(2) n'est autre que le rang du maximum suivant et ainsi de suite. Mais il y a un problème.

Lorsqu'on cherche le maximum d'un certain rang, il ne faut pas reprendre un maximum obtenu précédemment. Donc, lorsqu'on trouve un maximum, il faut remplacer l'élément par une valeur inférieure à toute valeur possible (ici -1) pour que l'élément ne soit plus repris. Dans ce cas, le tableau SM sera détruit par le classement. Pour l'éviter, on recopie d'abord le tableau SN sur lequel on travaillera.



D'où l'ordinogramme ci-contre qui n'est que la répétition de celui de 5.10 pour chaque valeur de K.

K représente le rang, à un instant donné, dans le classement.

```
250 DIM SN(NJ),PO(NJ)
260 FOR J=1 TO NJ:SN(J)=SM(J):NEXT
270 FOR K=1 TO NJ
280 JM=1:M=SN(JM)
290 FOR J=2 TO NJ
300 IF SN(J)<=M THEN 320
310 JM=J:M-SN(J)
320 NEXT J
330 PO(K)=JM:SN(JM)=-1
340 NEXT K
350 PRINT"CLASSEMENT":PRINT
360 PRINT"RANG","JOUEUR","SCORE MOYEN":PRINT
370 FOR I=1 TO NJ
380 PRINT I,NOM$(PO(I)),SM(PO(I)):NEXT I
```

Il existe d'autres méthodes de tri. Une des plus connues s'appelle méthode du "tri à bulles". On parcourt le tableau à classer en comparant à chaque fois deux éléments consécutifs. S'ils sont dans le bon sens, on les laisse ; s'ils sont dans le mauvais ordre, on les échange.

Si, lors d'un parcours, il y a eu au moins un échange, on fait un nouveau parcours. S'il n'y a eu aucun échange, c'est que, maintenant, le tableau est ordonné.

Les méthodes de classement s'appliquent au classement alphabétique des chaînes de caractères, puisque IF A$<B$ est vrai si A$ précède B$. La séquence suivante classe par ordre alphabétique le tableau des noms NOM$:

```
390 REM
400 ECH=0:REM  INDICATEUR D'ECHANGE
410 FOR I=1 TO NJ-1
420 IF NOM$(I)<=NOM$(I+1) THEN 450
430 T$=NOM$(I+1):REM SAUVEGARDE POUR ECHANGE
440 NOM$(I+1)=NOM$(I):NOM$(I)=T$:ECH=1
450 NEXT I
460 IF ECH=1 GOTO 400
470 PRINT:PRINT"CLASSEMENT ALPHABETIQUE"
480 FOR I=1 TO NJ:PRINT NOM$(I):NEXT I
```

Exercice 6.2

Deux problèmes se posent : le premier est celui de l'échelle ; les effectifs sont tous assez voisins (de 80 à 120), donc il faut dilater les différences, mais tout de même garder un certain terme constant.
Comme le numéro de la classe et l'effectif sont inscrits en tête de ligne, l'écriture commence en colonne 11. Il faut utiliser des manipulations de chaînes de caractères pour assurer une largeur constante aux impressions du numéro de classe et de l'effectif.

Nous proposons la formule de réduction d'échelle : $N=\frac{1}{2}(C(I)-70)$ qui fait varier N de 0 à 25.

D'où la fin du programme :

Exercice 6.2

```
90 PRINT "CL EFF"
100 FOR I=1 TO 10:N=INT((C(I)-70)/2)
110 PRINT RIGHT$("  "+STR$(I),3);" ";RIGHT$("  "+STR$(
                                              C(I)),4);
120 FOR K=1 TO N:PRINT "*";:NEXT K
130 PRINT:NEXT I
```

Exercice 6.3

```
10 INPUT "CENTRE(X,Y),RAYON";XC,YC,R
15 CLS
20 FOR T=0 TO 2*PI STEP 2*PI/200
30 X=INT(XC+R*COS(T)+.5)
40 Y=INT(YC+R*SIN(T)+.5)
50 PLOT X,Y,"*"
60 NEXT
```

Exercice 6.4

```
PLOT 14,13,CHR$(12)+CHR$(4)+CHR$(21)+"BONJOUR"+CHR$(23)
                 ↑         ↑         ↑                   ↑
       clignotement        |         |                   |
       caractères bleus              |      retour au fond blanc
               fond pourpre ─────────┘
```

Chaque attribut occupant une position, il faut commencer en colonne 14 pour que "bonjour" soit au milieu. Le CHR$(23) final fait revenir au fond blanc sinon le fond serait resté pourpre jusqu'à la fin de la ligne.

Exercice 6.5

```
PLOT 1,1Ø,"BONJOUR"+CHR$(12)+CHR$(7)+CHR$(16)+"MADAME"+
CHR$(23)+ CHR$(4)
PLOT 19,1Ø,CHR$(8)+"BONJOUR"+CHR$(12)+CHR$(3)+CHR$(2Ø)
+"MONSIEUR"
```

(On ne peut pas mettre tout en une seule ligne Basic).

Exercice 6.6

```
PLOT 17,13,2Ø:PLOT 18,13,23:PLOT 19,13,17:PLOT 2Ø,13,23
```

Exercice 6.7

```
1Ø CLS:PRINT CHR$(17)
2Ø FOR I=48ØØØ TO 4912Ø STEP 4Ø:POKE I,1Ø:NEXT
3Ø FOR I=48Ø14 TO 4912Ø STEP 4Ø:POKE I,23:NEXT
4Ø FOR I=48Ø28 TO 4912Ø STEP 4Ø:POKE I,17:NEXT
5Ø GOTO 5Ø
```

Au lieu de 5Ø, on pourrait faire retentir la Marseillaise, ce que nous saurons bientôt faire.

Le PRINT CHR$(17) en 1Ø n'est qu'un 'Ctrl' Q pour empêcher d'avoir le curseur.

Exercice 6.8

Il suffit d'ajouter les instructions :

```
90 A$="SINUSOIDE"
100 X=100:Y=191:CURSET X,Y,0:
110 FOR I=1 TO LEN(A$)
120 A=ASC(MID$(A$,I,1))
130 CHAR A,0,1
140 CURSET X+6*I,Y,0
150 NEXT
```

Exercice 6.9

```
      PROGRAMME  C-3
        TELECRAN

10 HIRES:PRINT"                TELECRAN"CHR$(17)
20 X=0:Y=0:CURSET X,Y,1
30 A=PEEK(520):B=PEEK(521):IF A=56 GOTO 30
40 IF A=188 THEN IX=1 :IY=0 :GOTO 110
50 IF A=156 THEN IX=0 :IY=-1:GOTO 110
60 IF A=172 THEN IX=-1:IY=0 :GOTO 110
70 IF A=180 THEN IX=0 :IY=1 :GOTO 110
80 IF A=132 THEN 10
90 IF A=175 THEN CURSET X,Y,0 :GOTO 20
100 GOTO 30
110 LX=(IX+1)*119.5:LY=(IY+1)*99.5:IF (X=LX) OR (Y=LY)
                                                GOTO 30
120 IF B=162 GOTO 140
130 CURSET X,Y,0
140 X=X+IX:Y=Y+IY:CURSET X,Y,1:GOTO 30
1000 PRINTPEEK(520):GOTO1000
```

L'impression de CHR$(17) en 1∅ supprime le curseur. Lorsque vous arrêtez de jouer par 'Ctrl' G, faites 'Ctrl' Q pour le rétablir. La ligne 11∅ assure qu'on ne sorte pas des limites de l'écran.

Par suite de l'effet de répétition des touches si vous voulez tracer un pointillé, vous ne pouvez pas maintenir l'appui sur la touche curseur et appuyer le doigt sur 'Ctrl' de façon intermittente : il faut aussi lever le doigt de la touche curseur.

Nous vous suggérons d'ajouter des touches pour faire les mouvements diagonaux.

Exercice 6.10

```
90 PRINT"ON RECOMMENCE(O/N)?"
91 GET A$
92 IF A$="O" GOTO 10
93 IF A$="N" THEN END
94 GOTO 91
```

Exercice 6.11

Trois solutions :

1- 1ØØ GET A$
   ... suite ...

2- 1ØØ A$=KEY$:IF A$=""GOTO 1ØØ

3- 1ØØ IF PEEK(52Ø)=56 GOTO 1ØØ

Exercice 7.1

Les seules instructions modifiées sont :

```
1Ø PRINT"  ";CHR$(27);"I          *"
2Ø PRINT"  ";CHR$(27);"I         * *"
3Ø G=471Ø4:R=ASC("*")
```

Le couple CHR$(27)(Escape)I fournit l'attribut 9. Les deux espaces en tête garantissent qu'il ne vienne pas effacer les attributs standard en tête de ligne.

Exercice 7.2

Il suffit d'ajouter

```
  5 CLS:?:?:?:?:?:?:?CHR$(17)
145 GOTO 145
```

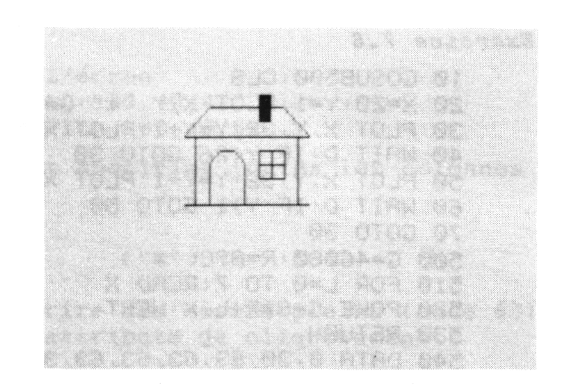
On sort du programme par 'Ctrl'C et on rétablit le curseur par 'Ctrl' Q.

Exercice 7.3

Il suffit d'ajouter un caractère plein en remplacement de K et □ en remplacement de L d'où le programme :

```
5 CLS:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT CHR$(17)
10 PRINT"             DDDDKD"
20 PRINT"            A    K B"
30 PRINT"           A        B"
40 PRINT"           HCCCCCCCCG"
50 PRINT"           F ACB GGEE"
60 PRINT"           F E F GGEE"
70 PRINT"           F E F CCLE"
80 PRINT"           JDIDJDDDDI"
90 G=47104
100 FOR R=65 TO 76
110 FOR L=0 TO 7
120 READ X:POKE G+8*R+L,X
130 NEXT L:NEXT R
140 FOR I=0 TO 26:PLOT 3,I,9:NEXT
145 GOTO145
150 DATA 1,2,2,4,8,16,16,32,32,16,16,8,4,2,2,1
160 DATA 63,0,0,0,0,0,0,0,0,0,0,0,0,0,0,63
170 DATA 32,32,32,32,32,32,32,32, 1,1,1,1,1,1,1,1
180 DATA 63,32,32,32,32,32,32,32,63,1,1,1,1,1,1,1
190 DATA 32,32,32,32,32,32,32,63, 1,1,1,1,1,1,1,63
200 DATA 63,63,63,63,63,63,63,63,32,0,0,0,0,0,0,0
```

La photo ci-contre donne l'aspect de l'écran obtenu.



Exercice 7.4

On utilise les remplacements des caractères de 'r' à 'z' mais dans le générateur standard, sinon l'écran serait troublé. Il n'y a donc pas d'attribut 9 à mettre.

```
     REM
2010 DATA &&&&&&6^,"%'/??___",^^____^^,?___??__,^_______,"_'''/#''"
2020 DATA ^^^^^^\\,"'#!     ","\\X0    "
2200 G=46080:FOR R=114 TO 122
2210 READ A$:FOR L=0 TO 7
2220 A=ASC(MID$(A$,L+1,1))-32
2230 POKE G+8*R+L,A:NEXT:NEXT
2300 CLS:PRINT:PRINT:PRINT:PRINT
2310 PRINT"           r"
2320 PRINT"          st"
2330 PRINT"          uv"
2340 PRINT"          wx"
2350 PRINT"          yz"
```

Exercice 7.5

```
10 GOSUB500:CLS
20 X=1:Y=13:PLOT X,Y,"*":D=6:S=1
30 PLOT X,Y,32:X=X+1:PLOT X,Y,"*"
40 WAIT D:IF X<38 GOTO 30
50 PLOT X,Y,32:X=X-1:PLOT X,Y,"*"
60 WAIT D:IF X>1 GOTO 50
70 D=D+S:IF D=10 THEN S=-1
80 IF D=1 THEN S=1
90 GOTO 30
500 G=46080:R=ASC("*")
510 FOR L=0 TO 7:READ X
520 POKE G+8*R+L,X:NEXT
530 RETURN
540 DATA 0,30,63,63,63,63,30,0
```

Exercice 7.6

```
10 GOSUB500:CLS
20 X=20:Y=1:PLOT X,Y,"*":D=6
30 PLOT X,Y,32:Y=Y+1:PLOT X,Y,"*"
40 WAIT D:IF Y<26 GOTO 30
50 PLOT X,Y,32:Y=Y-1:PLOT X,Y,"*"
60 WAIT D:IF Y>1 GOTO 50
70 GOTO 30
500 G=46080:R=ASC("*")
510 FOR L=0 TO 7:READ X
520 POKE G+8*R+L,X:NEXT
530 RETURN
540 DATA 0,30,63,63,63,63,30,0
```

Exercice 7.7

```
10 GOSUB500:CLS
20 X=1:Y=1:PLOT X,Y,"*":D=6
30 PLOT X,Y,32:Y=Y+1:X=X+1:PLOT X,Y,"*"
40 WAIT D:IF Y<26 GOTO 30
50 PLOT X,Y,32:Y=Y-1:X=X-1:PLOT X,Y,"*"
60 WAIT D:IF Y>1 GOTO 50
70 GOTO 30
500 G=46080:R=ASC("*")
510 FOR L=0 TO 7:READ X
520 POKE G+8*R+L,X:NEXT
530 RETURN
540 DATA 0,30,63,63,63,63,30,0
```

Exercice 7.8

Si $k \geqslant 32$, on a déjà $\ell 1=k$ et $\ell 2=k+64$.
$k'=63-k$ est un attribut. Le dessin complémentaire s'obtient par $\ell'=63-k+64=127-k$ d'où $\ell 3=\ell'+128=255-k$.

Exercice 7.9

C'est le piège ! Faites donc PAPER 3 et non
CURSET Ø,Ø,Ø:FILL 2ØØ,1,19

Exercice 7.10

PAPER 4:CURSET 8Ø,Ø,Ø:FILL 2ØØ,1,23:CURSET 16Ø,Ø,Ø:FILL 2ØØ,1,17

Exercice 7.11

```
1Ø HIRES:REM vide l'écran
2Ø CURSET Ø,Ø,Ø:FILL 1ØØ,4Ø,42
3Ø CURSET Ø,1ØØ,Ø:FILL 1ØØ,4Ø,54
```

Pour des motifs, il faut spécifier toutes les colonnes à remplir.

Exercice 7.12

Il faut cette fois inscrire des motifs pleins (code 63). Puis, en 7Ø on met les attributs de clignotement.

```
10 HIRES:PAPER 0:INK 4
20 FOR Y=1 TO 199
30 D=INT(0.5+14*SQR(1-(Y-100)^2/10000))
40 X1=6*(20-D):D2=2*D
50 CURSETX1,Y,0:FILL1,D2,63
60 NEXT
70 CURSET12,0,0:FILL200,1,12
```

Exercice 8.1

On joue le la 3 du diapason. 4Ø attend qu'on appuie sur une touche quelconque pour arrêter le son s'il est continu.

Exercice 8.2

```
1Ø MUSIC 1,3,1,Ø
2Ø MUSIC 2,3,5,Ø
3Ø MUSIC 3,3,8,Ø
4Ø PLAY 7,Ø,7,1
```

Exercice 8.3

On ajoute les instructions ci-dessous. 18Ø est modifié. NF est le tableau des fréquences, NO celui des octaves et ND celui des durées.

```
25 DATA3,4,3,4,3,4,8,4,8,4,10,4,10,4,3,5,12,4,8,4
26 DATA100,200,100,300,200,300,200,300,100,300
41 FOR N=1 TO 10:READ NF(N),NO(N):NEXT
42 FOR N=1 TO 10:READ ND(N):NEXT
180 CLS:PRINT"GAGNE":GOSUB 2000
2000 FOR N=1 TO 10
```

Exercice 8.3 *(suite)*

```
2010 MUSIC 1,NO(N),NF(N),10
2020 FOR Z=1 TO ND(N):NEXT
2030 MUSIC 1,NO(N),NF(N),0:NEXT
2040 PLAY0,0,0,0:RETURN
```

Exercice 8.4

JEU "SIMON"

```
10 DATA 1,3,5,6,8,10,12
20 FOR N=1 TO 7:READ NF(N):NEXT
30 CLS:PAPER 0:INK 7
40 FOR I=1 TO 4
50 NS(I)=INT(1+7*RND(1)):NEXT
60 FOR I=1 TO 4
70 PAPER NS(I)
80 MUSIC 1,4,NF(NS(I)),10
90 FOR Z=1 TO 400:NEXT
100 MUSIC 1,4,5,0:NEXT
110 T=0:PAPER 0
120 FOR I=1 TO 4
130 GET A$
140 SN(I)=ASC(A$)-48
150 MUSIC 1,4,NF(SN(I)),10:PAPER SN(I)
160 FOR Z=1 TO 400:NEXT
170 MUSIC 1,4,5,0:NEXT
180 BIEN=-1:PAPER 0:CLS
190 FOR I=1 TO 4
200 BIEN=BIEN AND (NS(I)=SN(I)) :NEXT
210 IF BIEN THEN PRINT "GAGNE":GOTO 240
220 IF T=0 THEN PRINT "REESSAYEZ":T=1:GOTO 120
230 PRINT "PERDU C'ETAIT ";
235 FOR I=1 TO 4:PRINTNS(I);:NEXT:PRINT
240 INPUT "ON RECOMMENCE ";A$
250 IF A$="OUI" GOTO 30
```

NS est la séquence préparée par l'ordinateur, tandis que SN est celle proposée par le joueur. Remarquez la variable logique BIEN et sa gestion. Bien entendu, vous pouvez apporter des variantes : nombre de notes par séquence et nombre de tentatives variables, possibilité de réentendre la séquence, traitement du cas où le joueur tape sur une autre touche que 1 à 7.

B I B L I O G R A P H I E

LIVRES

- Systèmes à Microprocesseurs
  par Daniel-Jean David (Editests)

- Programmer en Basic
  par Michel Plouin (Editions du P.S.I)

- Jeux, trucs et comptes pour PET/CEM
  par Michel Benelfoul (Editions du P.S.I)

- ORIC pour tous (ORIC-1 et ORIC-ATMOS)
  par Jacques Boisgontier et Sophie Brebion
  (Editions du P.S.I.)

- 52 programmes pour tous, ORIC-1 et ATMOS
  par Jacques Boisgontier (Editions du P.S.I.)

REVUES

- L'Ordinateur Individuel
  39, rue de la Grange aux Belles - 75010 PARIS

- La Commode, le magazine des ordinateurs Commodore, Atari et ORIC
  28, rue Vicq-d'Azir - 75010 PARIS

- MICR'ORIC

Achevé d'imprimer en mars 1984
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : mars 1984

N° d'impression : 402056
N° d'édition : 86595-109-3
ISBN : 2-86595-109-X



la découverte de l'ORIC

L'ORIC est un ordinateur individuel qui permet à la fois des applications sérieuses et les jeux. Ce livre d'initiation couvre les deux aspects. Il ne nécessite pas de connaissances préalables. Après une introduction formée de rappels généraux sur l'informatique, il comprend essentiellement une présentation progressive du langage Basic. La découverte du langage est conduite en bâtissant des programmes par améliorations successives au cours desquelles les notions nouvelles s'introduisent naturellement. On aborde spécialement les points forts de l'ORIC : graphiques, sons, couleurs, horloge. La délicate question des attributs est rendue compréhensible de façon très claire.

couverture A.GRAPH

EDITIONS DU P.S.I.
BP 86
77402 lagny-sur-marne cedex
ISBN 2.86595.109.X
imprimé en France


la découverte de l'ORIC
P.S.I.
daniel-jean david